

ATARI BASIC Handbuch

Julian Reschke

Atari

BASIC-Handbuch

Atari BASIC-Handbuch

Julian Reschke



BERKELEY · PARIS · DÜSSELDORF

Umschlagentwurf: Daniel Boucherie
Satz: tgr – typo-grafik-repro gmbh., remscheid
Gesamtherstellung: Druckerei Hub. Hoch, Düsseldorf

ISBN 3-88745-083-3
1. Auflage 1984

Printed in Germany

Inhaltsverzeichnis

Vorwort

Seit nunmehr gut fünf Jahren sind in Amerika Heimcomputer von Atari erhältlich. Damals setzten die Geräte 400 und 800 einen Leistungsstandard, den für lange Zeit kein anderer Heimcomputer erreichen konnte. Der Atari war der erste Heimcomputer, der mehr als Piepstöne erzeugen konnte und der über einen voll programmierbaren Graphikprozessor verfügte. Erst in jüngster Zeit gibt es Computersysteme, die mit den 256 Farben, den vier frei programmierbaren Tonkanälen, den frei bewegbaren Objekten (Players) etc. konkurrieren können.

Die Vorgänger der heutzutage erhältlichen XL-Modelle verfügten über kein eingebautes BASIC. Der Computer war vielmehr als „offenes" Gerät konzipiert wie viele größere Rechner. Inzwischen hat es der technische Fortschritt ermöglicht, daß das BASIC fest im Gehäuse eingebaut wird, ohne daß daraus irgendwelche Nachteile in bezug auf Nutzbarkeit und Erweiterbarkeit entstanden sind.

Das BASIC, das man heute in den XL-Computern findet, ist nur in wenigen Details (in erster Linie wurden Fehler korrigiert) verändert worden. Daher fehlen einige Befehle, die bei neu erschienenen Computern eigentlich standardmäßig vorhanden sind. Als Besitzer eines Atari-Computers ist man natürlich nicht an das eingebaute BASIC gebunden. Inwischen gibt es auch andere BASIC-Varianten, die einerseits mehr Befehle und andererseits auch eine größere Verarbeitungsgeschwindigkeit bieten (siehe unter dem Begriff BASIC). Aus diesem Grunde finden Sie in diesem Buch auch kurze Erläuterungen zu den Befehlen anderer BASIC-Varianten.

Neben der Beschreibung der BASIC-Wörter sind in diesem Buch auch Erklärungen anderer für den BASIC-Programmierer wichtige Begriffe enthalten.

In jedem Abschnitt, der ein BASIC-Wort erklärt, ist (neben der Angabe, in welchen BASIC-Varianten der Befehl verfügbar ist),
1. falls vorhanden, die Abkürzung
2. die Form des Befehls und
3. ein Beispiel für die Verwendung des Befehls angegeben. Zu manchen Begriffen ist zusätzlich ein Beispielprogramm abgedruckt, das den Umgang mit dem Befehl oder der Funktion erleichtern soll.

Ich hoffe, daß dieses Buch Ihnen die Verwendung von Atari-BASIC erleichtert und Ihnen vielleicht auch bei der Entscheidung zum Kauf einer anderen BASIC-Variante hilft.

Viel Spaß beim Programmieren!

A

Abkürzungen (Atari, BASIC XL)

In Atari-BASIC und BASIC XL können fast alle BASIC-Befehle abgekürzt werden. Dazu setzt man hinter den oder die ersten Buchstaben eines Befehls einen Punkt. So kann zum Beispiel der Befehl LIST durch „L." abgekürzt werden. In manchen Fällen ist es nicht eindeutig, als welcher Befehl eine Abkürzung erkannt wird. Die Abkürzung „G." könnte also sowohl GOTO, als auch GOSUB oder GET bedeuten. Bei der Erkennung dieser Abkürzungen geht das BASIC eine interne Tabelle aller Befehle durch, bis schließlich der erste passende Befehl eingesetzt wird. Da der Befehl REM der erste BASIC-Befehl in der internen Tabelle ist, wird ein einzelner Punkt („.") als REM interpretiert. Keine Regel ohne Ausnahme: Die anstelle von PRINT erlaubte Abkürzung „?" wird vom BASIC nicht in PRINT verwandelt, da das Fragezeichen in der internen Befehlstabelle einen eigenen Platz hat. Das Fragezeichen ist daher eigentlich keine Abkürzung für PRINT, sondern ein eigener Befehl, der aber exakt dieselbe Funktion hat. Weitere Beispiele für identische Befehle sind DIM/COM, GOTO/GO TO, DOS/CP (in BASIC XL) und die Wertzuweisung durch LET, bei der auch der Befehl weggelassen werden kann (→ Befehlssatz).

Bei den Erläuterungen zu den einzelnen BASIC-Befehlen (Ausnahme: Microsoft-BASIC-Befehle können nicht abgekürzt werden) wird immer die kürzest mögliche Abkürzung angegeben, wenn sich der Befehl abkürzen läßt. Außerdem wird hinter dem Befehlsnamen in Klammern angegeben, in welchen BASIC-Dialekten der betreffende Befehl verfügbar ist. Fehlt diese Angabe, dann existiert der Befehl in allen BASIC-Varianten.

ABS

Form: ABS(numausdr)
Beispiel: WERT=ABS(X)

Die Funktion ABS errechnet den Absolutwert einer Zahl. Das Argument der Funktion kann ein beliebiger numerischer Ausdruck, also beispielsweise eine Zahl, eine Variable oder irgendeine Funktion, die als Ergebnis einen numerischen Ausdruck liefert, sein.

Anmerkungen

- Die Funktion ABS wird überall da angewendet, wo das Vorzeichen einer Zahl bedeutungslos ist. So kann zum Beispiel mit ABS die Differenz zweier Zahlen unabhängig davon berechnet werden, welche von beiden größer ist.
- Zusammen mit SGN kann man diese Funktion benutzen, um eine Zahl in Betrag und Vorzeichen zu zerlegen.

```
10 REM ------------------------------
20 REM Demonstrationsprogramm zu ABS
30 REM ------------------------------
40 FOR I=1 TO 4:REM Das ganze vier Mal
50 REM 2 Zufallszahlen zwischen 0 und
10
60 ERSTEZAHL=RND(0)*10:ZWEITEZAHL=RND(
0)*10
70 REM Zahlen ausgeben
80 PRINT " Erste Zahl: ";ERSTEZAHL
90 PRINT "Zweite Zahl: ";ZWEITEZAHL
100 REM Differenz berechnen
110 PRINT "  Differenz: ";ABS(ERSTEZAH
L-ZWEITEZAHL)
120 PRINT :PRINT :REM 2 Leerzeilen
130 NEXT I
140 END
```

Abb. A.1: ABS – Beispielprogramm

```
  Erste Zahl: 9.51843261
 Zweite Zahl: 8.2849121
   Differenz: 1.23352051

  Erste Zahl: 1.87042236
 Zweite Zahl: 1.01638793
   Differenz: 0.85403443

  Erste Zahl: 5.53817749
 Zweite Zahl: 7.24777221
   Differenz: 1.70959472

  Erste Zahl: 8.36715698
 Zweite Zahl: 1.34307861
   Differenz: 7.02407837
```

Abb. A.2: ABS – Beispiellauf

ADR (Atari, BASIC XL)

Form: ADR(zausdr)
Beispiel: X=USR(ADR(CIO$))

Die Funktion ADR berechnet die Anfangsadresse einer Zeichenkette im Speicher des Computers. Als Argumente können sowohl Zeichenketten-Variablen als auch normale Zeichenketten zwischen Anführungsstrichen angegeben werden. Ist das Argument eine Variable (Beispiel: ADR(CIO$)), dann ist der Funktionswert die Adresse, an der diese Variable im Arbeitsspeicher abgelegt ist. Übergibt man als Argument eine gewöhnliche Zeichenkette (Beispiel: ADR("SYBEX")), dann ist der Funktionswert die Adresse, an der genau dieser Ausdruck im Listing des BASIC-Programms im Speicher steht.

Im Microsoft-BASIC heißt der entsprechende Befehl VARPTR.

Anmerkungen

- Die Funktion ADR wird meistens dazu benutzt, um die Anfangsadresse eines in einer Zeichenkette abgelegten Maschinenprogramms zu errechnen, um es dann mit USR(adr) aufrufen zu können. Es sollte beachtet werden, daß sich die Anfangsadresse einer Zeichenkette während des Programmablaufes ändern kann. Deshalb kann es zu Fehlern kommen, wenn mit Hilfe von ADR gleich zu Beginn diese Adresse berechnet und in einer Variablen abgelegt wird. Gerade bei USR-Aufrufen sollte die Anfangsadresse jedesmal neu berechnet werden. Nähere Informationen dazu unter „USR“.

```
10 REM ------------------------------
20 REM Demonstration zur Funktion ADR
30 REM ------------------------------
40 REM bisherige Programmzeile ausgebe
n
50 LIST 70
60 REM die folgende Zeile wird veraend
ert
70 LET ADRESSE=ADR("text")
80 POKE ADRESSE,ASC("T")
90 LIST 70:REM veraenderte Zeile ausge
ben
```

Abb. A.3: ADR – Beispielprogramm

```
70 LET ADRESSE=ADR("text")

70 LET ADRESSE=ADR("Text")
```

Abb. A.4: ADR – Beispiellauf

Adresse

Eine Adresse ist die Nummer eines Bytes im Speicher des Computers. Oft ist es sinnvoll, die Adresse eines bestimmten Speicherabschnitts zu kennen, zum Beispiel um den Bildspeicher abzuspeichern oder ein in einer Zeichenkette abgelegtes Maschinensprachprogramm aufzurufen.

AFTER (MSB)

Form: AFTER nausdr GOTO nausdr
Beispiel: AFTER 100 GOTO 1000

Wenn der Befehl AFTER auftritt, wird das erste Argument in ein internes Register übertragen, das daraufhin jede 1/50 Sekunde um eins verringert wird. Wird der Wert Null, dann springt das BASIC in die angegebene Zeilennummer.

Anmerkungen

- Der Befehl ist dann besonders sinnvoll einzusetzen, wenn nach einer festgesetzten Zeitspanne ein Programmteil ausgeführt werden soll.

Algorithmus

Ein Algorithmus ist eine Anleitung zur Lösung einer bestimmten Aufgabe. Um einen Computer programmieren zu können, muß zuerst ein allgemeiner Lösungsweg gefunden werden. Daraufhin müssen die Einzelschritte dieses Lösungswegs so weit detailliert werden, daß sie sich in für den Computer verständliche Befehle übersetzen lassen.

Beispiel: Es soll ein Programm geschrieben werden, mit dessen Hilfe man mit dem Joystick auf dem Bildschirm alle Farben darstellen und die dazugehörigen Farb- und Helligkeitswerte ablesen kann. Dieses allgemein beschriebene Problem kann der Computer nicht ohne eine

genaue Beschreibung der Einzelschritte lösen. Der folgende Algorithmus gibt schon eine genaue Beschreibung des Lösungswegs an, kann aber trotzdem noch nicht direkt in BASIC-Befehle übersetzt werden.

1. Wenn der Joystick nach oben gedrückt wird, dann erhöhe die Helligkeit um eine Stufe.
2. Wenn der Joystick nach unten gedrückt wird, dann verringere die Helligkeit um eine Stufe.
3. Wenn der Joystick nach rechts gedrückt wird, dann erhöhe den Farbwert um eine Stufe.
4. Wenn der Joystick nach links gedrückt wird, dann verringere den Farbwert um eine Stufe.
5. Überprüfe, ob die Werte noch innerhalb der erlaubten Grenzen liegen. Wenn nicht, dann korrigiere sie.
6. Setze die angegebene Farbe und gebe auf dem Bildschirm Farbe und Helligkeit aus.
7. Warte einen Moment.
8. Beginne wieder mit Anweisung Nr. 1.

Wenn Sie diesen Algorithmus mit dem fertigen BASIC-Programm vergleichen, dann werden Sie sehen, daß jeder dieser Schritte noch einmal in Einzelschritte zerlegt werden muß, um die Aufgabe für den Computer verständlich zu machen.

Um einen für den Computer verständlichen Algorithmus zu erstellen, müssen also die einzelnen Lösungsschritte so lange verfeinert werden, bis sie sich ungefähr in BASIC-Befehle übersetzen lassen.

```
10 REM ---------------------------
20 REM Demoprogramm zu Algorithmus
30 REM ---------------------------
40 FARBE=0:REM Variable fuer Farbwert
50 HELLIGKEIT=0:REM Variable fuer Hell
igkeit
60 REM je nach Joystickstellung Farbe
und Helligkeit veraendern
70 IF STICK(0)=14 THEN HELLIGKEIT=HELL
IGKEIT+2
80 IF STICK(0)=13 THEN HELLIGKEIT=HELL
IGKEIT-2
90 IF STICK(0)=7 THEN FARBE=FARBE+1
```

Abb. A.5: Algorithmus – Beispielprogramm

```
100 IF STICK(0)=11 THEN FARBE=FARBE-1
110 REM Werte ueberpruefen und korrigi
eren
120 IF FARBE>16 THEN FARBE=0
130 IF FARBE<0 THEN FARBE=15
140 IF HELLIGKEIT>14 THEN HELLIGKEIT=0

150 IF HELLIGKEIT<0 THEN HELLIGKEIT=14

160 REM Werte ausgeben
170 PRINT "Farbe: ";FARBE;"  Helligkei
t: ";HELLIGKEIT
180 REM Werte uebertragen
190 SETCOLOR 2,FARBE,HELLIGKEIT
200 REM einen Moment warten
210 FOR ZEIT=1 TO 10
220 NEXT ZEIT
230 REM und wieder von vorne
240 GOTO 60
```

Abb. A.5: Algorithmus – Beispielprogramm (Forts.)

```
Farbe: 8  Helligkeit: 14
Farbe: 9  Helligkeit: 14
Farbe: 10  Helligkeit: 14
Farbe: 11  Helligkeit: 14
Farbe: 11  Helligkeit: 14
Farbe: 12  Helligkeit: 14
Farbe: 13  Helligkeit: 14
Farbe: 13  Helligkeit: 0
Farbe: 13  Helligkeit: 0
Farbe: 13  Helligkeit: 0
Farbe: 14  Helligkeit: 0
Farbe: 15  Helligkeit: 0
Farbe: 16  Helligkeit: 0
Farbe: 16  Helligkeit: 0
Farbe: 16  Helligkeit: 2
Farbe: 0  Helligkeit: 2
Farbe: 1  Helligkeit: 2
Farbe: 1  Helligkeit: 4
Farbe: 2  Helligkeit: 4
Farbe: 3  Helligkeit: 4
Farbe: 3  Helligkeit: 4
Farbe: 3  Helligkeit: 4
Farbe: 4  Helligkeit: 4
Farbe: 4  Helligke
```

Abb. A.6: Algorithmus – Beispiellauf

Alphanumerisch

Unter alphanumerischen Ausdrücken versteht man Zeichenketten und Zeichenkettenvariablen. Zu den alphanumerischen Zeichen gehören eigentlich im engeren Sinne nur Buchstaben, Ziffern, Rechen- und Satzzeichen. Gewöhnlich zählt man aber auch die Graphik- und Sonderzeichen des normalen Zeichensatzes dazu.

AND

Form: logausdr AND logausdr
Beispiel: IF A>0 AND A<10 THEN PRINT "A liegt zwischen 1 und 9"

AND in BASIC

Der logische Operator AND verknüpft zwei logische Ausdrücke miteinander. Er wird dazu benutzt, um festzustellen, ob von zwei Bedingungen beide erfüllt sind. Ist auch nur eine der beiden durch AND verknüpften Bedingungen falsch, dann wird der gesamte Ausdruck falsch. Nur wenn beide Bedingungen erfüllt sind, wird der Gesamtausdruck zu einer wahren Aussage.

Nehmen wir einmal an, es sollte festgestellt werden, ob gleichzeitig der erste Joystick nach oben bewegt und der Schußknopf niedergedrückt wird. Dazu muß überprüft werden, ob einerseits STICK(0) den Wert 14 hat und andererseits die Funktion STRIG(0) den Wert 0 liefert. Die Bedingung muß deshalb IF STICK(0)=14 AND STRIG(0)=0 lauten.

Tabelle der Werte von AND:

1. Bedingung	2. Bedingung	Ergebnis
unwahr	unwahr	unwahr
unwahr	wahr	unwahr
wahr	unwahr	unwahr
wahr	wahr	wahr

Logisches AND

Neben der BASIC-Funktion AND existiert auch noch das „logische AND“, das vor allem in Maschinensprache häufig benutzt wird. Ähnlich wie beim BASIC-Befehl, wird auch hier das Ergebnis nur wahr, also „1“, wenn beide Einzelbedingungen, die verknüpft wurden, auch wahr sind. Im Gegensatz zum BASIC-AND werden jedoch beim logischen AND die einzelnen Bits der beiden Bytes unabhängig voneinander verglichen (das erste Bit der ersten Bytes mit dem ersten Bit des zweiten Bytes usw.).

Anführungsstriche (Atari, BASIC XL)

Anführungsstriche am Ende einer Zeichenkette dürfen dann weggelassen werden, wenn nach der Zeichenkette kein Befehl mehr folgt. Allerdings werden dann alle Kleinbuchstaben der Zeichenkette in Großbuchstaben verwandelt.

Append

Append bedeutet Anfügen von neuen Daten an eine bestehende Datei. Näheres dazu unter dem Befehl OPEN.

Argument

Ein Argument ist ein Zahlenwert oder eine Zeichenkette, die einer Funktion oder einem Unterprogramm zur Bearbeitung übergeben wird. So ist zum Beispiel bei ASC("Atari") die Zeichenkette "Atari" Argument der Funktion ASC.

Funktionen können auch mehrere Argumente haben. Ein Beispiel dafür ist die Funktion „USR“, bei der sogar praktisch beliebig viele Argumente übergeben werden dürfen.

ASC

Form: ASC(zausdr)

Beispiel: IF TASTE=ASC("J") THEN GOTO 1000

Mit der Funktion ASC wird der ATASCII-Wert (interner Zahlenwert für Zeichen) des ersten Zeichens einer Zeichenkette berechnet. Als Argument können sowohl eine Zeichenkettenvariable als auch eine Zeichenkette zwischen Anführungsstrichen angegeben werden. Ist die Zeichenkette länger als ein Zeichen, dann wird der ATASCII-Wert des ersten Zeichens berechnet. Übergibt man als Argument eine Zeichenkettenvariable mit der Länge „0", dann liefert ASC als Ergebnis den Wert „0".

Die Umkehrfunktion von ASC ist CHR$.

```
10 REM -----------------------------
20 REM Demonstrationsprogramm zu ASC
30 REM -----------------------------
40 REM Eingabezeichenkette definieren
50 DIM EINGABE$(40)
60 REM Zeichenkette eingeben
70 INPUT EINGABE$
80 REM von allen eingegebenen Zeichen
die ATASCII-Werte ausgeben
90 FOR I=1 TO LEN(EINGABE$)
100 PRINT "Zeichen: ";EINGABE$(I,I),"A
TASCII-Wert: ";ASC(EINGABE$(I))
110 NEXT I
120 REM neue Eingabe
130 GOTO 60
```

Abb. A.7: ASC – Beispielprogramm

```
?ATARI Basic-Handbuch
Zeichen: A          ATASCII-Wert: 65
Zeichen: T          ATASCII-Wert: 84
Zeichen: A          ATASCII-Wert: 65
Zeichen: R          ATASCII-Wert: 82
Zeichen: I          ATASCII-Wert: 73
Zeichen:            ATASCII-Wert: 32
Zeichen: B          ATASCII-Wert: 66
```

Abb. A.8: ASC – Beispiellauf

```
Zeichen: a          ATASCII-Wert: 97
Zeichen: s          ATASCII-Wert: 115
Zeichen: i          ATASCII-Wert: 105
Zeichen: c          ATASCII-Wert: 99
Zeichen: -          ATASCII-Wert: 45
Zeichen: H          ATASCII-Wert: 72
Zeichen: a          ATASCII-Wert: 97
Zeichen: n          ATASCII-Wert: 110
Zeichen: d          ATASCII-Wert: 100
Zeichen: b          ATASCII-Wert: 98
Zeichen: u          ATASCII-Wert: 117
Zeichen: c          ATASCII-Wert: 99
Zeichen: h          ATASCII-Wert: 104
?
```

Abb. A.8: ASC – Beispiellauf (Forts.)

Assembler

Siehe Maschinensprache.

ATASCII

Alle Buchstaben, Ziffern und Zeichen des Zeichensatzes werden im sogenannten ATASCII-Code abgespeichert. Im ATASCII-Code hat zum Beispiel das „A“ den Wert 65, das „B“ den Wert 66 etc. Die ATASCII-Zeichen 32 bis 127 stimmen im wesentlichen mit dem ASCII-Code (American Standard Code für Information Interchange) überein.

Abweichend davon werden die Zeichen im Bildschirmspeicher anders abgespeichert.

ATN

Form: ATN(numausdr)
Beispiel: WINKEL=ATN(TANGENS)

Die trigonometrische Funktion ATN berechnet den Arcustangens

(oder Arctangens) eines numerischen Ausdrucks. Je nachdem, ob der Rechner im Bogenmaß- oder Gradmodus rechnet, liegt das Ergebnis zwischen −PI/2 und PI/2 oder zwischen −180 Grad und 180 Grad. Es ist zu beachten, daß als Ergebnis nur der kleinste mögliche Winkel angegeben wird.

```
10 REM -----------------------------
20 REM Demonstrationsprogramm zu ATN
30 REM -----------------------------
40 REM Wertetabelle von 1 bis 9
50 PRINT "Argument","Ergebnis in"
60 PRINT "Bogenmass","Winkel"
70 PRINT "-----------------------------
--"
80 FOR ARGUMENT=1 TO 9
90 PRINT ARGUMENT
100 REM in Bogenmass
110 RAD :PRINT ATN(ARGUMENT),
120 REM in Grad
130 DEG :PRINT ATN(ARGUMENT)
140 NEXT ARGUMENT
```

Abb. A.9: ATN – Beispielprogramm

```
Argument  Ergebnis in
Bogenmass           Winkel
-------------------------------
1
0.7853981684        45.00000033
2
1.10714872          63.43494902
3
1.24904577          71.56505112
4
1.32581766          75.9637564
5
1.37340076          78.69006721
6
1.40564765          80.53767791
7
1.42889927          81.86989761
8
1.44644133          82.87498361
9
1.4601391           83.65980802
```

Abb. A.10: ATN – Beispiellauf

Atract-Modus

Um den Bildschirm zu schützen, beginnt der Atari nach ca. 5–7 Minuten nach dem letzten Tastendruck die Bildschirmfarben zu wechseln. Das kann bei Programmen, die z. B. nur durch Joystick-eingaben gesteuert werden, sehr lästig sein. Man kann diesen Effekt vermeiden, indem man in regelmäßigen Abständen den Befehl „POKE 77,0" durchführt.

Will man über die Tastatur den ATRACT-Modus abschalten, ohne daß das Programm den Tastendruck erkennt, so kann man dies zumeist durch Drücken von SHIFT-CTRL-A erreichen.

AUTO (MSB)

Form: AUTO nausdr,nausdr
Beispiel: AUTO 1000,10

Der AUTO-Befehl erleichtert die Erstellung von neuen Programmen ganz erheblich, indem er automatisch Zeilennummern ab einer bestimmten Anfangszeile (erstes Argument) mit einer bestimmten Schrittweite (zweites Argument) erzeugt.

In BASIC XL gibt es einen ähnlichen Befehl namens NUM.

B

BASIC

BASIC ist die Abkürzung für „Beginners All purpose Symbolic Instruction Code". Für den Atari gibt es momentan vier verschiedene BASIC-Varianten. Atari-BASIC wurde früher als Programmmodul verkauft und ist in alle neuen Atari-Computer eingebaut. Auch von Atari gibt es das Programmodul Microsoft-BASIC II, das vor allem bei der Benutzung von Programmen, die für andere Rechner geschrieben worden sind, Vorteile bietet. Auf einer dazugehörigen Ergänzungsdiskette gibt es nochmals eine Reihe interessanter zusätzlicher Befehle. Allerdings laufen in Atari-BASIC geschriebene Programme erst nach einer Anpassung verschiedener Befehle in Microsoft-BASIC. Von der Firma OSS gibt es BASIC XL, eine Weiterentwicklung des Standard-Atari-BASIC als Programmodul, die eine Reihe neuer Befehle im Bereich der Graphik und der strukturierten Programmierung bietet und gleichzeitig wesentlich schneller ist.

- Wenn sich einer der in diesem Buch beschriebenen Begriffe nur auf bestimmte BASIC-Dialekte bezieht, dann wird dies in Klammern hinter dem Begriff angegeben.

Befehlssatz

Der Befehlssatz des Atari-BASIC ist die Gesamtheit der BASIC-Befehle, die es versteht. Im Atari-BASIC ist eine Liste aller bekannten BASIC-Befehle abgespeichert, die zur Identifizierung der Befehle und ihrer Abkürzungen dienen (→ Abkürzungen). Das folgende Programm, das allerdings nur im Atari-BASIC läuft, gibt eine Liste aller BASIC-Befehle aus, mit der man auch die Rangordnung bei der Erkennung von Abkürzungen feststellen kann.

```
10 REM ------------------------
20 REM Demoprogramm Befehlssatz
30 REM ------------------------
40 LET LISTE=42161:REM Anfangsadresse
der internen Befehlsliste
50 NUMMER=1:PRINT "1. ";:REM Nummer de
s Befehls in der Liste
60 FOR BYTE=LISTE TO LISTE+335
70 WERT=PEEK(BYTE)
80 IF WERT<128 THEN PRINT CHR$(WERT);
90 IF WERT>=128 THEN PRINT CHR$(WERT-1
28):NUMMER=NUMMER+1:BYTE=BYTE+2:IF NUM
MER<55 THEN PRINT NUMMER;". ";
100 NEXT BYTE
```

Abb. B.1: Befehlssatz – Beispielprogramm

```
1. REM
2. DATA
3. INPUT
4. COLOR
5. LIST
6. ENTER
7. LET
8. IF
9. FOR
10. NEXT
11. GOTO
12. GO TO
13. GOSUB
14. TRAP
15. BYE
16. CONT
17. COM
18. CLOSE
19. CLR
20. DEG
21. DIM
22. END
23. NEW
24. OPEN
25. LOAD
26. SAVE
27. STATUS
28. NOTE
29. POINT
30. XIO
31. ON
32. POKE
33. PRINT
34. RAD
```

Abb. B.2: Befehlssatz – Beispiellauf

```
35. READ
36. RESTORE
37. RETURN
38. RUN
39. STOP
40. POP
41. ?
42. GET
43. PUT
44. GRAPHICS
45. PLOT
46. POSITION
47. DOS
48. DRAWTO
49. SETCOLOR
50. LOCATE
51. SOUND
52. LPRINT
53. CSAVE
54. CLOAD
```

Abb. B.2: Befehlssatz – Beispiellauf (Forts.)

Benutzerfreundlichkeit

Um Benutzerfreundlichkeit sollte sich jeder Programmierer bemühen. Ein benutzerfreundliches Programm sollte

- Programmabbrüche durch Fehler vermeiden, indem es Eingaben entweder so umwandelt, daß sie benutzt werden können, oder falsche Eingaben mit Begründung zurückweist (→ TRAP);
- mögliche Fehler des Benutzers (z. B. Formatieren einer wichtigen Diskette) vorhersehen und so unwahrscheinlich wie möglich machen;
- den Benutzer des Programms immer darüber informieren, was es von ihm erwartet oder was es gerade tut. Beispielsweise sollte es nicht vorkommen, daß eine längerdauernde Funktion ausgeführt wird (z. B. Sortieren), ohne daß der Benutzer weiß, welcher Programmteil gerade durchgeführt wird;
- graphische und akustische Möglichkeiten dazu einsetzen, Fehler zu vermeiden und die Bedienung zu erleichtern.

BGET (BASIC XL)

Abkürzung: BG.
Form: BGET #nausdr1 ,nausdr2 ,nausdr3
nausdr1: Kanalnummer
nausdr2: Adresse
nausdr3: Länge des Datenblocks
Beispiel: BGET#2,PLAYERS,2048

Der Befehl BGET lädt durchgehende Speicherbereiche aus dem durch den ersten Parameter angegebenen Kanal. Das zweite Argument gibt die Adresse, an die die Daten geladen werden sollen, das dritte die Länge des Speicherbereichs an.

Anmerkungen

- Da hier das Diskettenbetriebssystem eine schnellere Methode der Ein- und Ausgabe (die sog. Burst-IO) anwenden kann, wird der Befehl wesentlich schneller ausgeführt als etwa eine Schleife mit GET.

- Da das BASIC weder Anfangsadresse noch Länge der Daten überprüft, muß man selbst darauf achten, daß beim Laden keine anderen Daten, wie etwa das BASIC-Programm, zerstört werden.

Bildschirmdarstellung

Der Atari verfügt über einen voll programmierbaren Graphikprozessor, den ANTIC. Daher läßt sich die Bildschirmdarstellung beliebig aus verschieden großen Textbetriebsarten, hochauflösender Farbgraphik (→ GRAPHICS) und unabhängig vom Rest des Bildschirms bewegbaren Objekten (→ PLAYER-MISSILE-GRAPHIK) mischen. Neben den vom BASIC durch den Befehl GRAPHICS unterstützten Graphikstufen gibt es noch weitere Graphikmöglichkeiten, die nur unter Umgehung des Betriebssystems bei der Graphikprogrammierung benutzt werden können. Besonders in

Maschinensprachspielen wie HIGHWAY-DUEL, NADRAL oder CAVELORD, um nur zwei Beispiele zu nennen, werden diese für den NUR-BASIC-Programmierer brachliegenden Möglichkeiten genutzt.

Binäres Zahlensystem

Im binären Zahlensystem gibt es nur die Ziffern „0“ und „1“. Daher benötigt es zur Darstellung eines Bytes acht Stellen ($2^8=256$). Ein wesentlicher Vorteil der Binärdarstellung ist, daß der Zustand eines jeden Bits einfach festzustellen ist.

Bit

Das Bit ist die kleinste Informationseinheit des Computers. Man kann es sich als winzigen Schalter vorstellen, der nur zwei verschiedene Zustände einnehmen kann (1/0 oder ein/aus). Acht Bits bilden gemeinsam ein Byte (→ Byte).

Booten

Unter „Booten“ versteht man das automatische Laden des Programms durch Einschalten des Computers. Auf dem Atari muß dabei folgendes beachtet werden: Soll das Programm vom Kassettenrecorder geladen werden, dann muß während des Einschaltens die START-Taste gedrückt werden. Daneben ist es wichtig, ob es sich um ein Programm handelt, daß nur ohne eingelegtes oder eingeschaltetes BASIC funktioniert, was eigentlich in der Dokumentation angegeben sein müßte. Funktioniert das Programm nur ohne BASIC, muß auf den älteren Ataris vorher das BASIC-Modul entnommen werden oder auf den XL-Geräten während des Einschaltens die OPTION-Taste gedrückt werden.

BPUT (BASIC XL)

Abkürzung: BP.
Form: BPut#nausdr1 ,nausdr2 ,nausdr3
nausdr1: Kanalnummer
nausdr2: Adresse
nausdr3: Länge des Datenblocks
Beispiel: BPUT#1,FONT,1024

BPUT speichert den durch das zweite Argument adressierten Speicherbereich mit der durch Argument drei angegebenen Länge über den durch das erste Argument angegebenen Ein- und Ausgabekanal ab. Weitere Anmerkungen dazu unter BGET.

- Achtung: Aufgrund der Arbeitsweise des Diskettenbetriebssystems ist es nicht möglich, mit BPUT Speicherbereiche des ROM abzuspeichern.

BREAK-Taste

Das Drücken der BREAK-Taste während Eingaben ruft den Fehler 128 (BREAK-Taste gedrückt) hervor. Der Eingabemodus des Atari-BASIC reagiert einfach dadurch, daß er die laufende aktuelle Zeile verläßt und den Cursor eine Zeile tiefer setzt. Das Drücken der BREAK-Taste während der Programmausführung führt zum Abbruch des Programms. Dabei werden weder Variablen gelöscht noch verändert. Alle Tongeneratoren bleiben eingeschaltet. Das BASIC kehrt in die normale Textgraphikstufe zurück und gibt die Fehlermeldung „STOPPED AT LINE Zeilennummer" aus. Bei Eingabe von CONT wird das Programm bei der angegebenen Zeilennummer fortgesetzt.

Wird die BREAK-Taste während der Durchführung einer Ein- oder Ausgabeoperation gedrückt, wird der Vorgang angehalten bzw. fortgesetzt. Nur durch wiederholtes Drücken der BREAK-Taste kann das Programm während der Ein- und Ausgabe auf Drucker, Kassette oder Diskette unterbrochen werden. Aufgrund eines Fehlers im

Betriebssystem der Modelle 400/800 kann es dazu kommen, daß Ein- oder Ausgabevorgänge auf Diskette stoppen und erst nach einer Weile fortgesetzt werden. In einem solchen Fall bewirkt das Drükken der BREAK-Taste die Fortsetzung des Vorgangs.

- Es sollte möglichst daran gedacht werden, die BREAK-Taste vor Ein- und Ausgabevorgängen wieder anzuschalten, um Besitzern der älteren Geräte lästige Wartezeiten zu ersparen.

BUMP (BASIC XL)

Form: BUMP(nausdr1,nausdr2)
nausdr1: Nummer des einen Objekts
nausdr2: Numer des anderen Objekts
Beispiel: KOLLISION=BUMP (A,B)

Mit dieser Funktion können Kollisionen (Überlagerungen) zwischen Players, Missiles und Hintergrund festgestellt werden. Ist der Wert ungleich Null, dann hat eine Kollision stattgefunden.

Es können dabei Kollisionen zwischen Playern und Playern (Argument: 0–3,0–3), Missiles und Playern (Argument: 4–7,0–3), Playern und Hintergrund (Argument: 0–3,8–11) oder Missiles und Hintergrund (Argument: 4–7,8–11) abgefragt werden.

- Nachdem man auf Kollisionen überprüft hat, muß man die Kollisionsregister löschen. Zu diesem Zweck kann man POKE 53278,0 anwenden.

BYE (Atari, BASIC XL)

Abkürzung: B.
Form: BYE
Beispiel: BYE

Bei Eingabe dieses Befehls springt der Atari in eine Betriebssystemroutine an der Adresse $E471. Bei den Modellen 400/800 wird der sogenannte Blackboardmodus angesprungen, der sich mit der Mel-

dung „Atari-COMPUTER MEMO PAD“ meldet. In diesem Zustand reagiert der Atari zwar auf Tastendrücke, versteht aber keine Befehle und kann nur durch Drücken von SYSTEM-RESET oder durch Aus- und Wiedereinschalten initialisiert werden. Diese Betriebsart wurde eingebaut, damit sich die Modelle 400/800 auch ohne eingelegtes Programmodul und fehlender Bootmöglichkeit beim Einschalten melden. Die XL-Geräte springen nach Eingabe von BYE in das Selbsttestprogramm.

Byte

Das Byte ist die zweitkleinste Informationseinheit des Computerspeichers. Je nach Ausbaustufe verfügt der Atari über 16 K-Byte bis 64 K-Byte für den Anwender nutzbarer Speicher, wobei 1 K-Byte aus 1024 Bytes besteht.

Jedes Byte besteht aus acht Bits (→ Bit), so daß ein Byte genau 2^8 verschiedene Werte einnehmen kann und deshalb der Wert eines Bytes stets zwischen 0 und 255 liegt. Diese interne Struktur des Speichers erklärt, warum die Zahl 256 so oft im Zusammenhang mit Befehlen auftritt. So gibt es deshalb 256 verschiedene ATASCII-Zeichen (→ CHR$), weil jedes Zeichen in jeweils einem Byte abgespeichert wird.

Ein Byte kann ganz verschiedene Bedeutungen haben. So kann es natürlich einen einzelnen Wert darstellen, aber auch eine von vielen Stellen einer mehrstelligen Zahl sein. Genauso gut können seine einzelnen Bits darüber entscheiden, welche Pixels (→ Pixel) in hochauflösender Graphik (→ GRAPHICS) angeschaltet sind oder nicht.

C

CHR$

Form: CHR$(nausdr)
Beispiel: A$=CHR$(NUMMER)

Die Funktion CHR$ berechnet das zu einer Zahl dazugehörige ATASCII-Zeichen. Daher sind nur Argumente zwischen 0 und 255 sinnvoll, Werte bis zu 65535 werden jedoch dennoch angenommen.

- CHR$ ist die Umkehrfunktion zu ASC.

```
10 REM -----------------------------
20 REM Demonstrationsprogramm zu CHR$
30 REM -----------------------------
40 REM einen Kanal fuer die Tastatur o
effnen
50 OPEN #1,4,0,"K:"
60 REM Wert lesen
70 GET #1,TASTE
80 REM Zeichen und ATASCII-Code ausgeb
en
90 PRINT "Das Zeichen ";CHR$(TASTE);"
hat den Code ";TASTE;"."
100 GOTO 70
```

Abb. C.1: CHR$ – Beispielprogramm

```
Das Zeichen A hat den Code 65.
Das Zeichen T hat den Code 84.
Das Zeichen A hat den Code 65.
Das Zeichen R hat den Code 82.
Das Zeichen I hat den Code 73.
Das Zeichen   hat den Code 32.
Das Zeichen B hat den Code 66.
Das Zeichen A hat den Code 65.
```

Abb. C.2: CHR$ – Beispiellauf

```
Das Zeichen S hat den Code 83.
Das Zeichen I hat den Code 73.
Das Zeichen C hat den Code 67.
Das Zeichen - hat den Code 45.
Das Zeichen ^ hat den Code 94.
Das Zeichen a hat den Code 97.
Das Zeichen n hat den Code 110.
Das Zeichen d hat den Code 100.
Das Zeichen b hat den Code 98.
Das Zeichen u hat den Code 117.
Das Zeichen c hat den Code 99.
Das Zeichen h hat den Code 104.
Das Zeichen
 hat den Code 155.
```

Abb. C.2: CHR$ – Beispiellauf (Forts.)

CLEAR (MSB)

Identisch mit dem Atari-BASIC-Befehl CLR.

CLEAR STACK (MSB)

Form: CLEAR STACK
Beispiel: CLEAR STACK

Mit diesem Befehl kann der interne Speicher, der für AFTER, SOUND etc. benutzt wird, gelöscht werden. Dies kann nötig werden, wenn zu viele dieser Befehle kurz nacheinander ausgeführt worden sind, so daß Microsoft-BASIC nicht ausreichend internen Speicher hat, um sich alle Adressen und Zeiten zu merken.

CLOAD

Abkürzung: CLOA.
Form: CLOAD

Der Befehl CLOAD lädt ein mit CSAVE (→ CSAVE) auf Kassette gespeichertes BASIC-Programm wieder in den Computer ein.

Anmerkungen

- Als Ausgabekanal wird Kanal (→ IOCB) 7 benutzt. Wenn dieser schon für irgendeinen anderen Zweck belegt ist, dann wird er geschlossen und eine Fehlermeldung ausgegeben. Bei nochmaliger Eingabe von CLOAD sollte es daher keinen Fehler mehr geben.

Nach Eingabe des Befehls ertönt ein Brummton, der den Benutzer dazu auffordert, das Band mit REWIND und ADVANCE an die richtige Stelle zu spulen und PLAY zu drücken. Erst wenn dann irgendeine Taste außer den Funktionstasten und der BREAK-Taste gedrückt wird, beginnt das Band zu laufen. Die Datenübertragung zwischen Recorder und Computer wird durch typische Signaltöne begleitet, die über den Lautsprecher des Fernsehers übertragen werden.

Noch bevor der Ladevorgang beginnt, wird das im Speicher befindliche BASIC-Programm gelöscht. Auch wenn ein Fehler auftritt oder der Ladevorgang unterbrochen wird, ist das bisherige BASIC-Programm zerstört. Neben dem eigentlichen BASIC-Programm werden auch die BASIC-Statusvariablen und die Variablentabelle geladen. Das bedeutet, daß alle Variablen, die während der Programmerstellung benutzt worden sind, geladen werden, und zwar unabhängig davon, ob sie im Programm selbst auftauchen oder nicht. Wie man dieses Problem umgehen kann, wird unter LIST beschrieben.

Nach Ende des Ladevorgangs, egal ob erfolgreich, durch Fehler oder Abbruch, werden sämtliche Kanäle außer Kanal 0, der für die Bildschirmsteuerung benutzt wird, geschlossen. Das kann bei der Verwendung in anderen Graphikstufen (→ GRAPHICS) zu Fehlern führen, da alle Graphikstufen außer dem normalen Textmodus Kanal 6 benutzen (→ PRINT).

CLOG

Form: CLOG(nausdr)
Beispiel: A=PI*CLOG(1.1415)

Die Funktion CLOG berechnet den Logarithmus einer positiven

Zahl zur Basis 10. Der Logarithmus zu einer beliebigen Basis läßt sich mit der Formmel „CLOG(Argument)/CLOG(Basis)" ermitteln.

```
10 REM ---------------------
20 REM Demonstration zu CLOG
30 REM ---------------------
40 REM Einen Kanal fuer Editor oeffnen
50 CLOSE #1:OPEN #1,4,0,"E:"
60 REM Fehler abfangen
70 TRAP 180
80 REM Eingabe der beiden Parameter
90 PRINT "Logarithmus von ";
100 INPUT #1,ZAHL
110 PRINT "zur Basis ";
120 INPUT #1,BASIS
130 PRINT "Der Logarithmus von ";ZAHL
140 PRINT "zur Basis ";BASIS;" ist ";
150 PRINT CLOG(ZAHL)/CLOG(BASIS)
160 PRINT
170 GOTO 60
180 REM Fehler
190 PRINT CHR$(253);"nicht berechenbar
!"
200 PRINT
210 GOTO 60
```

Abb. C.3: CLOG – Beispielprogramm

```
Logarithmus von 10
zur Basis 2.4
Der Logarithmus von 10
zur Basis 2.4 ist 2.63011686

Logarithmus von 343
zur Basis 7
Der Logarithmus von 343
zur Basis 7 ist 3

Logarithmus von 100000
zur Basis 10

Der Logarithmus von 100000
zur Basis 10 ist 5

Logarithmus von 256
zur Basis 2
Der Logarithmus von 256
zur Basis 2 ist 7.99999997

Logarithmus von
```

Abb. C.4: CLOG – Beispiellauf

CLOSE

Abkürzung: CL.
Form: CLOSE#nausdr
Beispiel: CLOSE#6

Der CLOSE-Befehl schließt benutzte Datenkanäle. Als Argument muß die Nummer eines dem Benutzer zur Verfügung stehenden Kanals (1–7) übergeben werden. Ist der angegebene Kanal schon geschlossen, wird keine Fehlermeldung ausgegeben. Wenn er dagegen noch geöffnet ist, werden möglicherweise noch im Speicher befindliche Daten übertragen. Bei Diskettenbetrieb während der Ausgabe wird beispielsweise dann noch der letzte angefangene Sektor geschrieben und die Eintragung im Directory (Liste der auf der Diskette abgespeicherten Dateien) vorgenommen.

Anmerkungen

- Wenn ein offener Kanal mit OPEN geöffnet werden soll, kommt es zu einer Fehlermeldung. Daher sollte man überall dort, wo es nicht sicher ist, daß der betreffende Kanal schon geschlossen ist, dem OPEN-Befehl einen CLOSE-Befehl voranstellen.
- Bei Eingabe von END oder von DOS werden alle Kanäle außer Kanal 0 (Bildschirm) geschlossen.

```
10 REM ----------------------
20 REM Demonstration zu CLOSE
30 REM ----------------------
40 REM Datenkanal auf Diskette fuer
50 REM Ausgabe oeffnen
60 CLOSE #1:OPEN #1,8,0,"D:PROBE.DAT"
70 REM einige Zahlen in den Buffer
80 FOR I=1 TO 80
90 PUT #1,I
100 NEXT I
110 PRINT "Jetzt wird der Kanal geschl
ossen!"
120 CLOSE #1
```

Abb. C.5: CLOSE – Beispielprogramm

CLR (Atari, BASIC XL)

Abkürzung: nicht möglich
Form: CLR

Alle numerischen Variablen werden auf Null gesetzt. Zeichenkettenvariablen werden gelöscht und müssen für erneute Benutzung neu dimensioniert werden (→ DIM). Allerdings wird durch CLR nicht die Variablentabelle gelöscht, so daß sich die Anzahl der benutzten Variablen nicht verändert.

- Wie man eine Variablentabelle löschen kann, um Platz für neue Variablennamen zu gewinnen, wird unter LIST beschrieben. Außerdem wird der interne Zeiger für READ-Befehle wieder auf die erste DATA-Zeile zurückgesetzt (→ READ,DATA,RESTORE).

```
10 REM --------------------
20 REM Demonstration zu CLR
30 REM --------------------
40 REM Eine Zeichenkettenvariable
50 DIM TEXT$(40)
60 TEXT$="Atari Basichandbuch"
70 REM Eine Datenfeld
80 DIM DATEN(5)
90 FOR I=1 TO 5
100 DATEN(I)=I
110 NEXT I
120 REM Eine numerische Variable
130 NUMMER=1
140 REM Alle Variablen ausgeben
150 GOSUB 210
160 PRINT "Jezt wird der Befehl CLR au
sgefuehrt!"
170 CLR
180 REM Variablen wieder ausgeben
190 GOSUB 210
200 END
210 REM Unterprogramm zur Ausgabe der
220 REM Variablen
230 PRINT NUMMER
240 REM Hier wird nach CLR ein Fehler
250 REM auftreten, weil die Dimensio-
260 REM nierung nicht wiederholt wor-
270 REM den ist
280 FOR I=1 TO 5
290 PRINT DATEN(I)
300 NEXT I
310 PRINT TEXT$
320 RETURN
```

Abb. C.6: CLR – Beispielprogramm

```
1
1
2
3
4
5
Atari Basichandbuch
Jezt wird der Befehl CLR ausgefuehrt!
0

ERROR-    9 AT LINE 290
```

Abb. C.7: CLR – Beispiellauf

CLS (MSB)

Form: CLS nausdr
Beispiel: CLS 200

Löscht den Bildschirminhalt und überträgt die angegebene Farbe in das Farbregister für den Bildrand.

COLOR

Abkürzung: C.
Form: COLOR nausdr
Beispiel: COLOR FARBE

Der Befehl COLOR gibt an, welche Farbe oder welches Zeichen beim folgendem PLOT- oder DRAWTO-Befehl benutzt wird. Als Argument sind Zahlen zwischen 0 und 65535 zulässig, obwohl die größte sinnvolle Zahl 255 ist.

Da der Befehl COLOR in den verschiedenen Graphikstufen verschiedene Wirkungen hat, wird er für alle verschiedenen Gruppen von Graphikarten einzeln erklärt.

COLOR in GRAPHICS 0

Im normalen Textmodus gibt COLOR den ATASCII-Wert des Zeichens an, das bei der Benutzung von PLOT oder DRAWTO erscheinen wird. GRAPHICS 0 ist also gleichzeitig eine Graphikstufe mit einer Auflösung von 24 * 40 Zeichen und 256 verschiedenen „Zeichenfarben“.

```
100 REM -------------------
110 REM COLOR IN GRAPHICS 0
120 REM -------------------
130 REM BILDSCHIRM INITIALISIEREN
140 CLOSE #6:OPEN #6,12,0,"S:"
150 REM MIT STERNCHEN
160 COLOR ASC("*")
170 REM RAHMEN MALEN
180 PLOT 0,0
190 DRAWTO 0,23
200 DRAWTO 39,23
210 DRAWTO 39,0
220 DRAWTO 0,0:PLOT 0,0
230 POSITION 5,10:PRINT "Fertig ist de
r Rahmen!"
```

Abb. C.8: COLOR in GRAPHICS 0 – Beispielprogramm

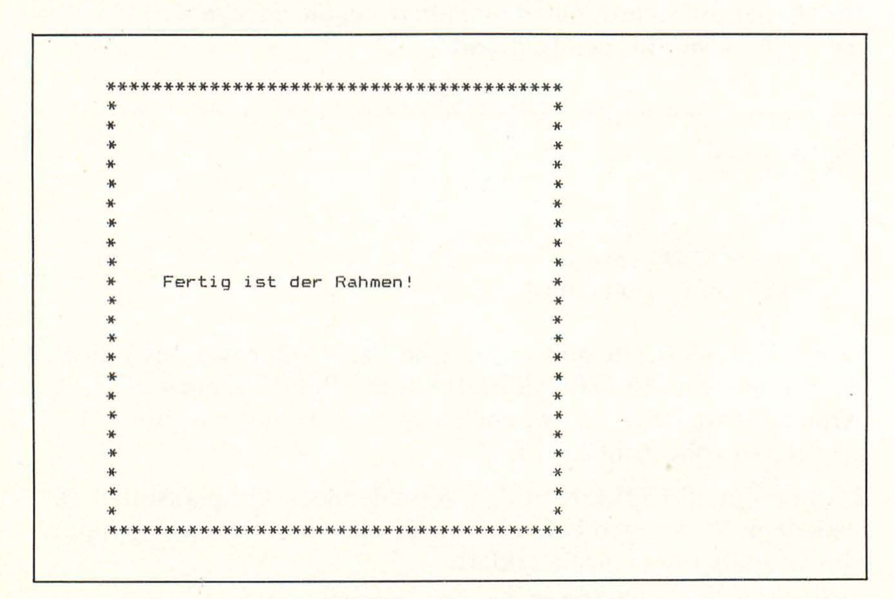


Abb. C.9: COLOR in GRAPHICS 0 – Beispiellauf

COLOR in GRAPHICS 1 und 2

Auch in den Graphikmodi 1 und 2 wird durch COLOR die Nummer eines Graphikzeichens festgelegt.

Da in diesen Graphikstufen jedoch nur 64 verschiedene Zeichen existieren, kann durch mehrfache Addition von 64 eine von vier möglichen Farben ausgewählt werden. Die zweite Hälfte des Zeichensatzes, mit dem die ATASCII-Zeichen von 0 bis 31 und von 96 bis 127 dargestellt werden können, kann durch POKE 756,226 eingeschaltet werden. Den Standardzeichensatz, der die ATASCII-Zeichen von 32 bis 95 enthält, erhält man dann wieder durch POKE 756,224.

```
100 REM -------------------------
110 REM COLOR IN GRAPHICS 1 UND 2
120 REM ---------------------------
130 REM GRAPHIKSTUFE EINSCHALTEN
140 GRAPHICS 1
150 PRINT "Graphics 1 mit Zeichensatz
1"
160 REM SPALTENWEISE LINIEN ZIEHEN
170 FOR I=0 TO 19
180 COLOR I+1+64*INT(RND(0)*4)
190 PLOT I,0:DRAWTO I,19
200 NEXT I
210 GOSUB 410
220 PRINT "Graphics 1 mit Zeichensatz
2"
230 REM ZEICHENSATZ 2 EINSCHALTEN
240 POKE 756,226
250 GOSUB 410
260 REM ALTEN ZEICHENSATZ EINSCHALTEN
270 POKE 756,224
280 REM IN GRAPHICS 2
290 GRAPHICS 2
300 PRINT "Graphics 2 mit Zeichensatz
2"
310 FOR I=0 TO 19
320 COLOR I+1+64*INT(RND(0)*4)
330 PLOT I,0:DRAWTO I,9
340 NEXT I
350 GOSUB 410
360 REM ZEICHENSATZ 2 EINSCHALTEN
370 POKE 756,226
380 GOSUB 410
390 REM GRAPHIK AUSSCHALTEN
400 GRAPHICS 0:END
410 REM TASTENDRUCK ABWARTEN
420 CLOSE #1:OPEN #1,4,0,"K:"
430 GET #1,TASTE
440 RETURN
```

Abb. C.10: COLOR in GRAPHICS 1 und 2 – Beispielprogramm

COLOR in GRAPHICS 3 bis 7, 14 und 15

In diesen Graphikstufen stehen vier Zeichenfarben zur Verfügung. Einen Sonderfall bilden die Graphikstufen 4 und 6, in denen nur zwei Zeichenfarben vorhanden sind, und die deshalb auch nur die Hälfte des Speichers belegen. Mit COLOR wird festgelegt, welches der Farbregister die Farbe des nächsten gesetzten Punktes festlegen soll (→ SETCOLOR). Welche der Farben der COLOR-Befehl einschaltet, geht aus der folgenden Tabelle hervor.

Nummer des Farbregsiters bei SETCOLOR	Argument von COLOR in den Graphikstufen 4,6	3,5,7,14,15
0	1	1
1	–	2
2	–	3
3	0	0 (Hintergrundfarbe)

```
100 REM ----------------------------
110 REM COLOR IN GRAPHICS 3-7,14,15
120 REM ----------------------------
130 REM Graphikstufe einschalten
140 GRAPHICS 7+16
150 SETCOLOR 0,4,6
160 SETCOLOR 1,8,8
170 SETCOLOR 2,12,10
180 DEG :REM in Grad
190 FOR COL=1 TO 3:REM in drei Farben
200 COLOR COL
210 FOR RADIUS=10 TO 20
220 FOR WINKEL=(COL-1)*120 TO COL*120
STEP 10
230 PLOT 80-COS(WINKEL)*RADIUS,48-SIN(
WINKEL)*RADIUS
240 NEXT WINKEL
250 NEXT RADIUS
260 NEXT COL
270 REM Bild eingeschaltet lassen
280 GOTO 270
```

Abb. C.11: COLOR in GRAPHICS 3 bis 7, 14 und 15 – Beispielprogramm

COLOR in GRAPHICS 8

In der höchstauflösenden Graphikstufe steht nur eine Farbe mit zwei Helligkeiten zur Verfügung. Ist das Argument 0, dann wird in der Hintergrundfarbe gemalt. Ist es 1, dann wird ein Punkt in der Hintergrundfarbe mit der von Farbregister 2 definierten Helligkeit gemalt.

Nummer des Farbregisters	Argument von COLOR in Graphikstufe 8
0	nicht benutzt
1 (Helligkeit)	1
2	0
3	nicht benutzt
4	nicht benutzt

- Gibt man als Farbwert den ATASCII-Wert des „Clear"-Zeichens (125) an und führt danach einen PLOT-Befehl durch, wird der Bildschirminhalt gelöscht.

```
100 REM -------------------
110 REM COLOR IN GRAPHICS 8
120 REM -------------------
130 REM Graphikstufe einschalten
140 GRAPHICS 8+16
150 REM schwarzer Hintergrund,Farbe 1
160 SETCOLOR 2,0,0:COLOR 1
170 REM in Grad
180 DEG
190 REM etwas malen
200 FOR WINKEL=0 TO 120 STEP 10
210 PLOT 160-80*SIN(WINKEL),96-80*COS(
WINKEL)
220 DRAWTO 160-80*SIN(WINKEL+120),96-8
0*COS(WINKEL+120)
230 DRAWTO 160-80*SIN(WINKEL+240),96-8
0*COS(WINKEL+240)
240 DRAWTO 160-80*SIN(WINKEL+360),96-8
0*COS(WINKEL+360)
250 DRAWTO 160-80*SIN(WINKEL),96-80*CO
S(WINKEL)
260 NEXT WINKEL
270 REM Anhalten
280 GOTO 270
```

Abb. C.12: COLOR in GRAPHICS 8 – Beispielprogramm

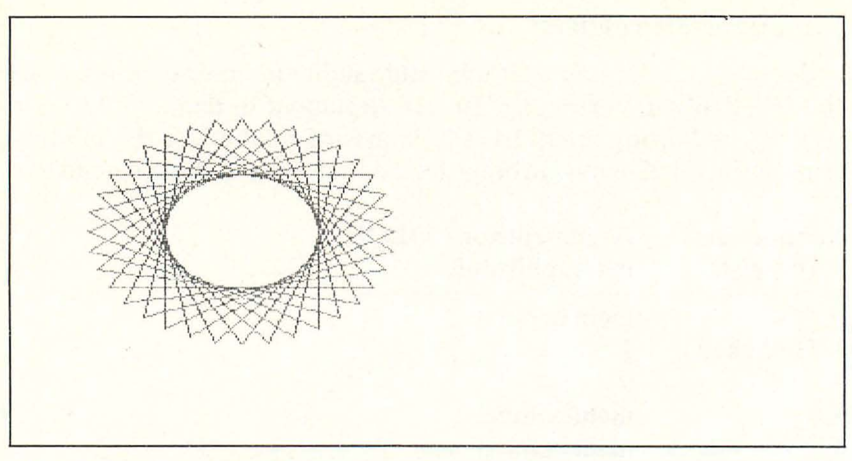


Abb. C.13: COLOR in GRAPHICS 8 – Beispiellauf

COLOR in GRAPHICS 9, 10 und 11

Die Graphikstufen 9 bis 11 sind die sogenannten GTIA-Modi, da sie mit Hilfe des GTIAs (Georges Television Interface Adapter) erzeugt werden. Ältere amerikanische Geräte verfügen nur über die ältere Version dieses Graphikchips, den CTIA, und können diese drei Graphikstufen nicht ansteuern.

In den Graphikstufen 9 und 11 stehen 16 verschiedene Zeichenfarben zur Verfügung, während es in Graphikstufe 10 nur 9 verschiedene Farben sind. In den ersteren sind jedoch die Farben voneinander abhängig: In GRAPHICS 9 kann man nur 16 Farbtöne einer Farbe benutzen, in GRAPHICS 11 nur 16 verschiedene Farbtöne gleicher Helligkeit.

In Graphikstufe 9 wird die Zeichenfarbe durch Farbregister 4 festgelegt. Alle Zeichenregister haben diese Farbe, wobei Farbe 0 die dunkelste und Farbe 15 die hellste der Farben ist.

Auch in GRAPHICS 11 hat das Farbregister 4 Einfluß auf alle anderen Zeichenfarben. Alle Zeichenfarben haben die in diesem Register angegebene Helligkeit, während der Farbton durch die zwischen 0 und 15 liegende Zeichenfarbe ausgewählt wird.

Für GRAPHICS 10 gilt folgende Tabelle:

Speicherstelle	Argument von COLOR
704	0 (Hintergrund)
705	1
706	2
707	3
708	4
709	5
710	6
711	7
712	8

- Die Speicherstellen 708 bis 712 sind identisch mit den Farbregistern 0 bis 4. Wie man die anderen Farben beeinflußt, wird unter SETCOLOR erklärt.

```
100 REM ------------------
110 REM COLOR IN GRAPHICS 9
120 REM ------------------
130 GRAPHICS 9
140 GOSUB 180
150 REM neue Hintergrundfarbe
160 SETCOLOR 4,RND(0)*16,0
170 GOTO 140
180 REM Vertikal
190 ANF=RND(0)*63
200 FOR SPALTE=ANF TO ANF+15
210 COL=2*(SPALTE-ANF):IF COL>15 THEN
COL=31-COL
220 COLOR COL
230 PLOT SPALTE,0:DRAWTO SPALTE,191
240 NEXT SPALTE
250 REM Horizontal
260 ANF=RND(0)*159
270 FOR ZEILE=ANF TO ANF+31
280 COL=ZEILE-ANF:IF COL>15 THEN COL=3
1-COL
290 COLOR COL
300 PLOT 0,ZEILE:DRAWTO 79,ZEILE
310 NEXT ZEILE
320 RETURN
```

Abb. C.14: COLOR in GRAPHICS 9 – Beispielprogramm

```
100 REM --------------------
110 REM COLOR IN GRAPHICS 10
120 REM --------------------
130 GRAPHICS 10
140 REM Farben initialisieren
150 FOR I=1 TO 8
160 POKE 704+I,(I-1)*32+8
170 NEXT I
180 REM in Grad
190 DEG
200 FOR RADIUS=0 TO 16
210 COLOR RADIUS/2
220 FOR WINKEL=0 TO 359 STEP 10
230 DRAWTO 40-RADIUS*COS(WINKEL),96-RA
DIUS*SIN(WINKEL)
240 NEXT WINKEL
250 NEXT RADIUS
260 REM Farben wechseln
270 OFFSET=8
280 FOR I=1 TO 8
290 FARBE=(I-1)*32+OFFSET
300 IF FARBE>255 THEN FARBE=FARBE-256:
GOTO 300
310 POKE 705+I,FARBE
320 NEXT I
330 OFFSET=OFFSET+16
340 REM Warten
350 FOR ZEIT=1 TO 50
360 NEXT ZEIT
370 GOTO 280
```

Abb. C.15: COLOR in GRAPHICS 10 – Beispielprogramm

COLOR in GRAPHICS 12 und 13

Diese Graphikstufen sind eigentlich Textstufen, in denen die einzelnen Zeichen aus verschiedenen Farben bestehen dürfen. Genau wie in GRAPHICS 0 kann man zwischen 256 verschiedenen Zeichen auswählen. Durch die Verdoppelung der Anzahl der Farben ist die Auflösung innerhalb der Zeichen halbiert. Daher kann der Standardzeichensatz in diesen Graphikstufen nicht benutzt werden. Liegt das Argument über 128, wird also ein inverses Zeichen benutzt, dann kann eine fünfte Farbe benutzt werden.

Da in diesen Graphikstufen jeder Bildpunkt durch zwei Bits angesprochen wird, kann man durch verschiedene Kombinationen der Bits aus den verschiedenen Farben auswählen.

Nummer des Farbregisters	Bitkombination innerhalb des Zeichens
0	01
1	10
2	11 (normales Zeichen)
3	11 (inverses Zeichen)
4	00

Anmerkungen

- Die Graphikstufen 12 bis 15 sind auf den Geräten 400/800 nicht durch BASIC-Befehle ansprechbar.

COM (Atari, BASIC XL)

Abkürzung: nicht möglich
Form: COM zeichenkettenvariable(nausdr)
COM numvariable(nausdr)
Beispiel: COM FILENAME$(17)

COM stellt Speicherplatz für Zeichenkettenbariblen und Fehler von numerischen Variablen zur Verfügung. COM ist identisch mit dem gebräuchlicheren Befehl DIM.

COMMON (MSB)

Form: COMMON varname,....
Beispiel: COMMON A,B,Z

Durch den Befehl COMMON kann man bestimmen, daß die dahinter angegebenen Variablen ihre Werte behalten, wenn das nächste Programm geladen wird. Gibt man stattdessen ALL an, dann werden alle Variablen übernommen.

Anmerkungen

- Dieser Befehl ist besonders dann nützlich, wenn man innerhalb eines BASIC-Programms mit RUN „Dateiname" ein anderes Programm aufrufen will, ohne daß die Variablen ihre Werte verlieren.

CONT

Abkürzung: CON.
Form: CONT

Mit Hilfe des CONT-Befehls kann ein Programm nach einer Unterbrechung fortgesetzt werden. Dabei beginnt die Programmausführung mit der auf die laufende Zeile folgenden Programmzeile. Das heißt, daß zum Beispiel beim Auftreten eines Fehlers nach Eingabe von CONT das Programm in der darauffolgenden Zeile fortgesetzt wird. Ist das Programm überhaupt noch nicht gestartet worden, dann beginnt die Programmausführung mit der zweiten Zeile.

Wenn das Programm wegen eines Fehlers unterbrochen wurde, dann sollte nicht CONT benutzt werden, da sonst möglicherweise für das Programm wichtige Befehle übersprungen werden. Es ist auf jeden Fall besser, die Fehlerquelle zu beseitigen und die Programmausführung mit GOTO bei der laufenden Zeilennummer fortzusetzen.

CP (BASIC XL)

Form: CP

Der Befehl CP (Control Program) ist identisch mit DOS.

COS

Form: COS(nausdr)
Beispiel: LET COSINUS=COS(WINKEL)

Die Funktion COS berechnet den Kosinus. Je nachdem, ob sich der Rechner im Grad- oder im Bogenmaßmodus (→ DEG, RAD) befindet, ist das Argument in Grad oder Bogenmaß anzugeben.

```
10 REM --------------------
20 REM Demonstration zu COS
30 REM --------------------
40 REM in Grad rechnen
50 DEG
60 REM Hochaufloesende Graphik
70 GRAPHICS 8+16
80 REM Schwarzer Hintergrund
90 SETCOLOR 2,0,0
100 REM In Weiss malen
110 COLOR 1
120 REM Kosinuskurve malen
130 REM Koordinatenkreuz malen
140 PLOT 0,96:DRAWTO 319,96:PLOT 159,0
:DRAWTO 159,191:PLOT 0,0
150 REM Kurve malen
160 FOR WINKEL=0 TO 359 STEP 5
170 DRAWTO WINKEL*8/9,96-COS(WINKEL)*9
5
180 NEXT WINKEL
190 REM Bild angeschaltet lassen, bis
200 REM BREAK gedrueckt wird
210 GOTO 210
```

Abb. C.16: COS – Beispielprogramm

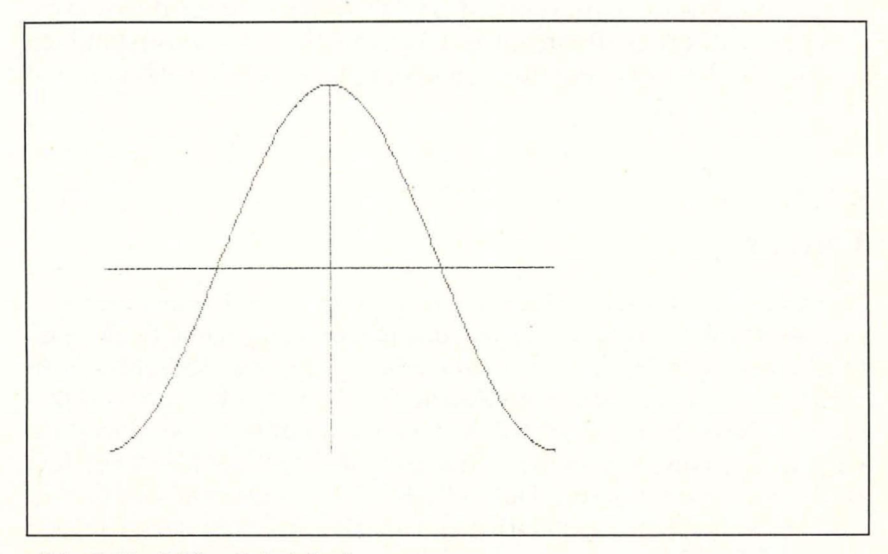


Abb. C.17: COS – Beispiellauf

CSAVE

Abkürzung: CS.
Form: CSAVE
Beispiel: CSAVE

Der Befehl CSAVE speichert das im Speicher befindliche BASIC-Programm auf Kassette ab. Dazu wird zunächst Kanal 7 für Ausgabe auf Kassette geöffnet. Tritt dabei ein Fehler auf, dann wird Kanal 7 geschlossen und ein Fehler angezeigt. Bei nochmaliger Eingabe von CSAVE sollte es daher nicht mehr zu Fehlern kommen.

Als nächstes wird ein Brummton ertönen, um den Benutzer dazu aufzufordern, das Band an die richtige Stelle zu spulen und irgendeine Taste außer den Funktionstasten und der Breaktaste zu drükken. Danach beginnt dann der Aufzeichnungsvorgang.

Anmerkungen

- Mit CSAVE auf Kassette abgespeicherte Programme können nur mit CLOAD wieder geladen werden.
- Die Ausführung des Befehls läßt sich durch wiederholtes Drücken der Breaktaste, Drücken von SYSTEM-Reset oder durch Ausschalten des Kassettenrecorders unterbrechen. Selbstverständlich läßt sich das so bei der Abspeicherung unterbrochene Programm nicht mehr laden.

Cursor

Der Cursor ist im Atari-BASIC ein weißes Feld auf blauem Grund, das die Position anzeigt, an der das nächste eingegebene Zeichen erscheinen wird. Wird der Cursor über irgendeine Bildschirmposition bewegt, an der schon ein Zeichen vorhanden ist, dann wird dieses invertiert. Der Cursor läßt sich mit bestimmten Tasten wie den Cursortasten und TAB sowie durch Befehle wie POSITION auf dem Bildschirm positionieren. Durch POKE 752,1 kann man den Cursor unsichtbar machen. Nach Drücken von BREAK oder nach Eingabe von POKE 752,0 wird er wieder sichtbar.

Ein Cursor kann jedoch auch anders aussehen. Bei manchen anderen Computern blinkt er beispielsweise. Daneben kann er auch in anderen Programmen auf dem Atari ein anderes Aussehen haben. So ist der Cursor in dem Textverarbeitungsprogramm „Atari-Schreiber“ eine blinkende Linie, die sich unter dem betreffenden Zeichen befindet.



DATA

Abkürzung: D.
Form: DATA konstante, konstante,...
Beispiel: DATA BASIC, Pascal, Action, Assembler
DATA 1,2,3,4,5,6,7,8,9

Mit dem Befehl DATA werden Konstanten festgelegt, die später mit dem READ-Befehl Variablen als Wert zugewiesen werden. Die Einschränkung auf Konstanten hat zur Folge, daß jeder Ausdruck, der nicht als Zahl erkannt werden kann, als Zeichenkette gilt. Es ist also

```
100 REM ---------------------
110 REM Demonstration zu DATA
120 REM ---------------------
130 DATA ABS,ASC,CHR$,ATN,CLOG
140 DATA 1,2,3,4,5,6,7,8,9,10
150 REM Zeichenkettenvariable definier
en
160 DIM TEXT$(4)
170 REM Datenfeld definieren
180 DIM ZAHL(10)
190 REM Zeichenketten lesen und ausgeb
en
200 FOR I=1 TO 5
210 READ TEXT$:PRINT TEXT$
220 NEXT I
230 REM Zahlen lesen und ausgeben
240 FOR I=1 TO 10
250 REM hinter READ duerfen keine
260 REM Feldvariablen stehen!
270 READ WERT
280 REM Daher erst in andere Variable
290 REM lesen und dann uebertragen
300 ZAHL(I)=WERT
310 PRINT ZAHL(I)
320 NEXT I
```

Abb. D.1: DATA – Beispielprogramm

weder möglich, Terme wie „645/868686“ anzugeben, noch irgendeine Variable einzusetzen. Schreibt man zum Beispiel „DATA A$,NUMMER“, dann werden beim READ-Befehl die Zeichenketten “A$“ und “NUMMER“ gelesen. Zwei aufeinanderfolgende Leerzeichen werden als Zeichenkette mit der Länge Null interpretiert (→ READ, RESTORE).

```
ABS
ASC
CHR$
ATN
CLOG
1
2
3
4
5
6
7
8
9
10
```

Abb. D.2: DATA – Beispiellauf

Datei

Eine Datei ist eine Sammlung von Informationen, die auf irgendein Peripheriegerät geschrieben oder von ihm gelesen werden können. Diskettendateien können auch teilweise gelesen oder verändert werden (→ PUT, GET, OPEN, CLOSE, PRINT, INPUT etc). Jede Datei hat eine Dateispezifikation, die aus dem Gerätenamen und gegebenenfalls einem Dateinamen besteht. Kassettendateien haben daher immer die Dateispezifikation “C:“. Bei Diskettendateien kann zusätzlich noch ein aus acht Buchstaben bestehender Name und eine aus drei Buchstaben bestehende Namenserweiterung angegeben werden, die den Dateityp angibt. Ein typischer Dateiname ist zum Beispiel “D1:BEISPIEL.EXT“ (wenn mit Diskettenlaufwerk eins gearbeitet wird, darf übrigens die „1“ weggelassen werden).

Anmerkungen

- Um zu gewährleisten, daß auch Dritte Dateinamen verstehen können, ist es sinnvoll, sich an bestimmte Namenserweiterungen zu halten, die sich eingebürgert haben.

Namenserweiterung	Dateityp
SYS	Systemdateien wie DOS und DUP
BAS	Mit SAVE abgespeicherte BASIC-Programme
LST	Mit LIST abgespeicherte BASIC-Programme
ASM	Mit LIST abgespeicherte Assemblerprogramme
M65	Mit SAVE abgespeicherte MAC65-Dateien
DOC	Textdateien
COM oder CMD	Maschinenprogramme
OBJ	Maschinenprogramme, die nicht in der vorliegenden Form gestartet werden dürfen
DAT	Dateien mit Daten
EXC	OS/A+ Executedateien
304 (Versionsnummer)	Mit SAVE abgespeichertes BASIC A+ Programm
ACT	Action!-Programmtexte

DEF (MSB)

Format: DEF name(variable,variable)=funktionsausdruck
Beispiel: DEF LN(ZAHL,BASIS)=LOG(ZAHL)/LOG(BASIS)

Mit dem Befehl DEF können eigene BASIC- Funktionen definiert sein. Dabei wird hinter dem Befehl DEF der Name der zu definierenden Funktion mit den in Klammern stehenden benutzten Variablen, ein Gleichheitszeichen und der Funktionsausdruck angegeben.

Neben numerischen Variablen können auch Zeichenkettenvariablen übergeben werden.

DEFDBL (MSB)

Format: DEFDBL variablenname, variablenname-variablenname
Beispiel: DEFDBL A,Y–Z

Mit DEFDBL wird festgelegt, daß die angegebenen Variablen in doppelter Genauigkeit bearbeitet werden sollen. Die Benutzung des Befehls entspricht der des Befehls DEFSGN.

DEFINT (MSB)

Format: DEFINT variablenname, variablenname-variablenname
Beispiel: DEFINT Q,W–Z

Mit diesem Befehl werden Variablen als ganzzahlig festgelegt. Der Befehl wird genauso benutzt wie DEFSNG.

DEFSGN (MSB)

Format: DEFSGN variablenname, variablenname-variablenname
Beispiel: DEFSGN A,E–G

Mit dem Befehl DEFSGN kann man festlegen, daß bestimmmte Variablen nur in einfacher Genauigkeit abgespeichert werden sollen, was sowohl Platz als auch Zeit spart.

Im obigen Beispiel werden alle Variablen, deren Namen mit „A" anfangen oder deren erster Buchstabe zwischen „E" und „G" liegt, in einfacher Genauigkeit bearbeitet.

DEFSTR (MSB)

Format: DEFSTR variablenname, variablenname-variablenname
Beispiel: DEFSTR A,E–G

Mit DEFSTR kann eine Variable, auch ohne Dollarzeichen („$") als letztes Zeichen, als String deklariert werden. Ansonsten arbeitet der Befehl wie DEFSNG.

DEG (Atari, BASIC XL)

Abkürzung: DE.
Form: DEG

Dieser Befehl schaltet in den Gradmodus um. Das BASIC wird nach Eingabe dieses Befehls alle Argumente von trigonometrischen Funktionen als Winkelwerte in Grad interpretieren.

Nach dem Einschalten und nach SYSTEM-Reset befindet sich der Rechner im Bogenmaßmodus. Daher sollte der DEG-Befehl innerhalb des Programms stehen, um Fehler zu vermeiden.

```
100 REM --------------------
110 REM Demonstration zu DEG
120 REM --------------------
130 REM Kanal fuer Bildschirm oeffnen
140 CLOSE #1:OPEN #1,12,0,"E:"
150 REM Zahl eingeben
160 PRINT "Wert in Bogenmass: ";
170 INPUT #1,BMASS
180 REM Auf Bogenmass schalten
190 RAD
200 SINUS=SIN(BMASS)
210 REM Auf Grad zurueckschalten
220 DEG
230 REM Gradwert berechnen
240 GRAD=ATN(SINUS/SQR(-SINUS*SINUS+1)
)
250 REM Ergebnis ausgeben
260 PRINT BMASS;" = ";GRAD;" Grad."
270 GOTO 160
```

Abb. D.3: DEG – Beispielprogramm

```
Wert in Bogenmass: 1.54
1.54 = 88.23549482 Grad.
Wert in Bogenmass: 0.1
0.1 = 5.72957796 Grad.
Wert in Bogenmass: 0.2
0.2 = 11.45915589 Grad.
Wert in Bogenmass: 0.3
0.3 = 17.18873386 Grad.
Wert in Bogenmass: 0.40
0.4 = 22.91831178 Grad.
Wert in Bogenmass: 0.5
0.5 = 28.64788977 Grad.
Wert in Bogenmass: 0.6
0.6 = 34.37746781 Grad.
Wert in Bogenmass: 0.7
0.7 = 40.1070458 Grad.
Wert in Bogenmass: 0.8
0.8 = 45.83662374 Grad.
Wert in Bogenmass:
```

Abb. D.4: DEG – Beispiellauf

DEL (BASIC XL, MSB)

Abkürzung: nicht möglich
Form: DEL startnummer-endnummer (MSB)
DEL startnummer,endnummer (BASIC XL)

Der Befehl DEL löscht die Programmzeilen, die zwischen den angegebenen Zeilennummern liegen.

Delete

Löschen einer Datei auf der Diskette durch Eingabe des betreffenden DOS-Befehls oder von BASIC aus durch „XIO 33,#kanalnummer,0,0,dateiname“.

DIM

Abkürzung: DL.
Form: DIM variable(nausdr),variable(nausdr),...
Beispiel: DIM DAT$(128),TABELLE(10,10)

Der DIM-Befehl stellt Speicherplatz für Zeichenkettenvariablen oder Datenfelder zur Verfügung. Als Argument muß zuerst der Name der Variablen und dann innerhalb der Klammern entweder die Länge der Zeichenkette oder bei Datenfeldern die Anzahl der Variablen in jeder Dimension angegeben werden. Datenfelder dürfen ein- oder zweidimensional sein.

Mit dem DIM-Befehl wird die größtmögliche Länge von Zeichenkettenvariablen eingestellt. Bei der Dimensionierung ist zu beachten, daß jedes einzelne Zeichen einer Zeichenkettenvariablen nur ein Byte belegt, so daß es viel Speicherplatz sparen kann, wenn man Datenfelder durch Zeichenketten ersetzt.

Bei der Dimensionierung von Datenfeldern wird für jedes Element ein Speicherplatz von sechs Bytes reserviert.

Wird eine Variable neu definiert, ohne daß dazwischen ein CLR-Befehl ausgeführt wurde, wird ein Fehler auftreten.

- BASIC XL bietet außerdem die Möglichkeit, Zeichenkettenfelder zu dimensionieren. Daneben ist es in BASIC XL nicht nötig, eindimensionale Zeichenketten vor Gebrauch zu dimensionieren, sofern sie eine vorher festgelegte Länge nicht überschreiten werden.

```
100 REM --------------------
110 REM Demonstration zu DIM
120 REM --------------------
130 REM Zeichenketten dimensionieren
140 DIM EINGABE$(6):REM fuer verschied
ene Zwecke
150 DIM PARTEIEN$(30)
160 REM Zeichenketten definieren
170 PARTEIEN$="Union SPD    F.D.P.Gruen
eSonst."
180 REM Datenfelder dimensionieren
190 DIM JAHR(2),PROZENT(2,5)
200 REM Bildschirm inittialisieren
```

Abb. D.5: DIM – Beispielprogramm

```
210 E=1:CLOSE #E:OPEN #E,12,0,"E:"
220 POKE 752,0:REM Cursor sichtbar
230 REM Menu ausgeben
240 POSITION 10,2:PRINT "Wahldiagramm"

250 POSITION 5,5:PRINT "1. Jahreszahle
n eingeben."
260 POSITION 5,7:PRINT "2. Anteile im
Jahr ";JAHR(1)
270 POSITION 5,9:PRINT "3. Anteile im
Jahr ";JAHR(2)
280 POSITION 5,11:PRINT "4. Graphische
 Darstellung."
290 POSITION 5,14:PRINT "->";
300 INPUT #E,BEFEHL
310 ON BEFEHL GOTO 340,430,480,530
320 REM Eingabefehler
330 GOTO 200
340 REM Jahrezahlen eingeben
350 PRINT CHR$(125):REM Bild loeschen
360 POSITION 10,2:PRINT "Jahreszahlen"

370 PRINT
380 PRINT "Letzte Wahl: ";
390 INPUT #E,WERT:JAHR(1)=WERT
400 PRINT " Diese Wahl: ";
410 INPUT #E,WERT:JAHR(2)=WERT
420 GOTO 200
430 REM Werte der letzten Wahl
440 PRINT CHR$(125)
450 POSITION 5,2:PRINT "Anteile bei de
r letzten Wahl"
460 NUMMER=1:GOSUB 710
470 GOTO 200
480 REM Werte der jetzigen Wahl
490 PRINT CHR$(125)
500 POSITION 7,2:PRINT "Anteile bei di
eser Wahl"
510 NUMMER=2:GOSUB 710
520 GOTO 200
530 REM Graphische Darstellung
540 GRAPHICS 7
550 REM Cursor ausschalten
560 POKE 752,1
570 REM Farben einschalten
580 SETCOLOR 0,2,12:SETCOLOR 1,8,8:SET
COLOR 2,3,14
590 FOR PARTEI=1 TO 5
600 PRINT PARTEIEN$(PARTEI*6-5,PARTEI*
6);"  ";
610 FOR JAHR=0 TO 1
620 COLOR JAHR+1
630 HOEHE=80-PROZENT(JAHR+1,PARTEI)*4/
5
640 FOR SPALTE=PARTEI*32-(JAHR+1)*14 T
O PARTEI*32-1-(JAHR)*8
650 PLOT SPALTE,80:DRAWTO SPALTE,HOEHE

660 NEXT SPALTE
670 NEXT JAHR
```

Abb. D.5: DIM – Beispielprogramm

```
680 NEXT PARTEI
690 PRINT
700 PRINT "Taste druecken";:GET #E,WER
T:GOTO 200
710 REM Werte eingeben
720 PRINT
730 FOR PARTEI=1 TO 5
740 PRINT PARTEIEN$(PARTEI*6-5,PARTEI*
6);": ";
750 INPUT #E,WERT:PROZENT(NUMMER,PARTE
I)=WERT
760 NEXT PARTEI
770 RETURN
```

Abb. D.5: DIM – Beispielprogramm (Forts.)

```
      Wahldiagramm

1. Jahreszahlen eingeben.

2. Anteile im Jahr 0

3. Anteile im Jahr 0

4. Graphische Darstellung.

->


      Wahldiagramm

1. Jahreszahlen eingeben.

2. Anteile im Jahr 1980

3. Anteile im Jahr 1984

4. Graphische Darstellung.

->
```

Abb. D.6: DIM – Beispiellauf

DIR (BASIC XL)

Form: DIR zausdr
Beispiel: DIR
DIR "D2:*.BAS"

Der Befehl DIR dient zur Ausgabe des Directorys auf dem Bildschirm. Als Parameter muß der Dateiname der gesuchten Dateien in „Wild-Card"-Form stehen. Um alle Dateien mit der Namenserweiterung „.BAS" in Laufwerk zwei zu ermitteln, müßte man beispielsweise als Argument "D2:*.BAS" angeben. Läßt man das Argument weg, werden alle Dateien, die sich auf der Diskette in Laufwerk eins befinden, ausgegeben.

Directory

Das Directory ist die Liste der auf der Diskette gespeicherten Programme. Es befindet sich in allen DOS-Versionen außer Atari-DOS 3.0 in den Sektoren 369 bis 376 (den Sektoren in der Mitte der Diskette), um eine möglichst kurze Zugriffszeit zu gewährleisten. Weitere Informationen über das Directory unter OPEN.

```
100 REM ---------------------------
110 REM Demonstration zu Directory
120 REM ---------------------------
130 DIM DATEI$(20)
140 REM Kanal fuer Laden aus dem
150 REM Directory oeffnen
160 CLOSE #1:OPEN #1,6,0,"D:*.*"
170 REM Fehler bei EndEnde abfangen
180 TRAP 230
190 REM Eintrag holen
200 INPUT #1,DATEI$
210 PRINT DATEI$
220 GOTO 200
230 REM Ende des Directorys
240 PRINT
250 END
```

Abb. D.7: Directory – Beispielprogramm

```
DOS        SYS 039
WDP        SYS 042
A          AW  093
B          AW  078
C          AW  111
D          AW  087
E          AW  106
F          AW  067
069 FREE SECTORS
```

Abb. D.8: Directory – Beispiellauf

Direkt auszuführender Befehl

Ein direkt auszuführender Befehl ist ein BASIC-Befehl, der ohne Voranstellung einer Zeilennummer im Eingabemodus eingetippt wurde. Mittels Trennung durch Doppelpunkte ist es möglich, auch mehrere BASIC-Befehle hintereinander ausführen zu lassen. Die meisten BASIC-Befehle können sowohl als direkter Befehl als auch im Programm verwendet werden.

Wird eine Datei mit ENTER geladen, dann wird jede einzelne Zeile als direkt eingetipptes Kommando verstanden. So kann man, indem man an ein mit LIST abgespeichertes BASIC-Programm eine Datei mit dem Inhalt „RUN" anhängt, einen Autostart installieren. Dazu müssen nur folgende Befehle ausgeführt werden:

```
OPEN #1,9,0, name des BASIC-Programms
PRINT #1, "RUN"
CLOSE #1
```

Wenn man dieses Programm mit ENTER lädt, wird es sich selbsttätig starten.

DOS

Abkürzung: DO.
Form: DOS
Beispiel: DOS

Der Befehl DOS ruft das Diskettenbetriebssystem auf, nachdem alle

offenen Kanäle geschlossen worden sind. Bei Arbeit unter dem Atari-Diskettenbetriebssystem 2.0s wird das Disk Utility Package nachgeladen. Das im Speicher befindliche BASIC-Programm geht nur dann nicht verloren, wenn auf der Diskette die MEM.SAV-Datei angelegt worden ist. Arbeitet man unter OS/A+ oder unter Atari DOS 1.0, dann wechselt der Computer nur in das DOS, ohne etwas dazuzuladen. Dafür müssen zu besonderen Zwecken Hilfsprogramme nachgeladen werden, was unter Atari-Betriebssystemen nicht nötig ist. Ist kein DOS geladen worden, so verhält sich der Computer wie bei Eingabe des Befehls BYE.

- In BASIC XL gibt es zusätzlich den Befehl CP (Control Program), der die gleiche Wirkung hat.

DPEEK (BASIC XL)

Form: DPEEK(nausdr)
nausdr: Abzufragendes Doppelbyte
Beispiel: SCREEN=DPEEK(88)

Ähnlich wie PEEK dient DPEEK der Abfrage des Wertes einzelner Bytes. Allerdings wird mit DPEEK der Inhalt von Zwei-Byte-Registern abgefragt, so daß das Ergebnis zwischen 0 und 65535 liegen kann.

- Bei Zwei-Byte-Registern ist das zweite Byte das höherwertige (Most Significant Byte – MSB). Daher müßte man ohne DPEEK folgendermaßen vorgehen: PEEK (adresse)+256*PEEK (adresse+1).

DPOKE (BASIC XL)

Abkürzung: DP.
Form: DPOKE nausdr1,nausdr2
nausdr1: Adresse
nausdr2: Neuer Inhalt
Beispiel: DPOKE DLIST,1536

Der Befehl DPOKE wird verwendet, um den Wert eines Zwei-Byte-Registers im Speicher zu verändern. Das erste Argument bestimmt dabei die Adresse und das zweite den neuen Wert des Doppelbytes (→ DPEEK, POKE).

DRAWTO (Atari, BASIC XL)

Abkürzung: DR.
Form: DRAWTO nausdr1,nausdr2
nausdr1: X-Position
nausdr2: Y-Position
Beispiel: DRAWTO SPALTE,ZEILE

Der Befehl DRAWTO zeichnet vom letzten gesetzten Punkt eine Linie zu der angegebenen Bildschirmposition. Diese Position kann nicht durch PUT, PRINT usw. verändert werden. Als Zeichenfarbe wird die beim letzten COLOR-Befehl angegebene Zeichenfarbe benutzt (→ COLOR). Je nach Anzahl der Bildpunkte in der eingeschalteten Graphikstufe wird die gezeichnete Linie mehr oder weniger gestuft aussehen.

```
100 REM -----------------------
110 REM Demonstration zu DRAWTO
120 REM -----------------------
130 GRAPHICS 8+16
140 SETCOLOR 2,0,0:COLOR 1
150 REM Stern malen
160 FOR I=0 TO 319 STEP 5
170 PLOT I,0:DRAWTO 319-I,191
180 NEXT I
190 FOR I=191 TO 0 STEP -3
200 PLOT 0,I:DRAWTO 319,191-I
210 NEXT I
220 COLOR 0
230 REM Warten
240 GOTO 230
```

Abb. D.9: DRAWTO – Beispielprogramm

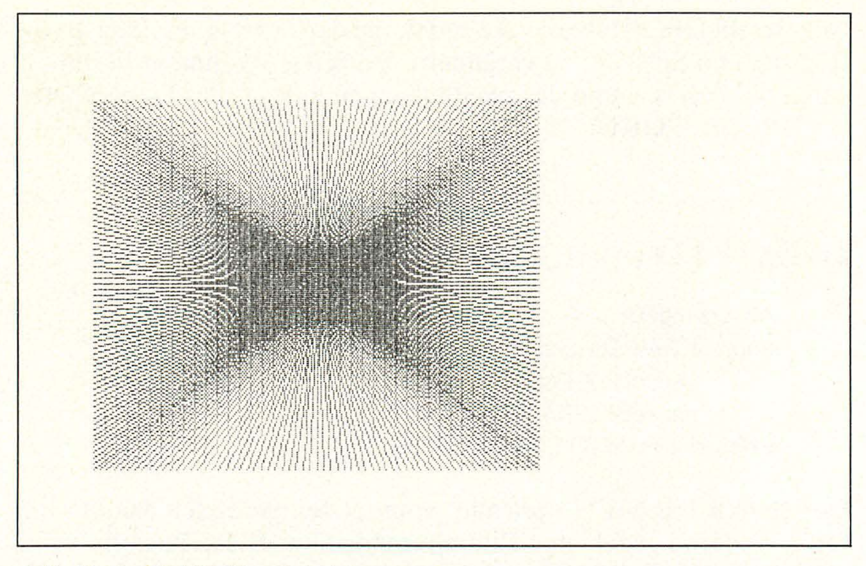


Abb. D.10: DRAWTO – Beispielprogramm

Dualsystem

Siehe binäres Zahlensystem.

Dummy

Eine Dummy-Variable ist eine Variable, die zwar aus irgendwelchen Gründen benutzt werden muß (z. B. als Argument bei RND), die aber ansonsten keine Bedeutung für den Programmablauf hat.



Editor

Der Editor ist jener Teil des Betriebssystems, der Ein- und Ausgaben auf den Bildschirm steuert.

END

Abkürzung: nicht möglich
Form: END
Beispiel: END

Der Befehl END beendet die Ausführung des BASIC-Programms. Alle Tongeneratoren werden ausgeschaltet, alle Kanäle bis auf Kanal 0 (Bildschirm) werden geschlossen und der normale Textmodus wird aktiviert. Soll die Programmausführung mit der letzten Zeile des Programms aufhören, dann kann der END-Befehl weggelassen werden, da das BASIC automatisch einen END-Befehl durchführt, wenn es keine Programmzeile mehr findet.

ENDWHILE (BASIC XL)

Abkürzung: ENDW.
Form: ENDWHILE
Beispiel: ENDWHILE

Beendet eine WHILE-Schleife (siehe auch WHILE).

ENTER (Atari, BASIC XL)

Abkürzung: E.
Form: ENTER zausdr
zausdr: Gerätename des Peripheriegeräts
Beispiel: ENTER "D:PROGRAMM.LST
ENTER "C:

Der ENTER-Befehl lädt ein mit dem LIST-Befehl abgespeichertes BASIC-Programm ein. Mit SAVE oder CSAVE gespeicherte Programme können mit ENTER nicht geladen werden. Als Argument muß die Dateispezifikation des zu ladenden Programms angegeben werden. Das ist bei Laden von Kassette einfach nur "C:", während bei Diskettenbetrieb noch zusätzlich der Programmname und die Namenserweiterung angegeben werden muß (z. B. "D2:BEISPIEL.LST").

Bei der Ausführung des ENTER-Befehls werden nacheinander die einzelnen Zeilen der Datei geladen. Bei der Abspeicherung der Zeilen geht das BASIC so vor, als wäre jede Folge, daß das im Speicher befindliche Programm von dem zu ladenden Programm nicht überschrieben (also gelöscht), sondern ergänzt wird. Man kann also mit ENTER mehrere BASIC-Programme zusammenhängen, wobei man jedoch aufpassen sollte, daß sich die Zeilennummern der einzelnen Programmzeilen nicht überschneiden. Ist in dem Programm, das geladen wird, eine Zeile mit bereits vorhandener Zeilennummer, so wird die im Speicher befindliche Programmzeile gelöscht. Neben Programmzeilen kann eine Datei, die mit ENTER geladen wird, auch direkte Befehle enthalten, die in dem Moment ausgeführt werden, in dem sie geladen sind.

Der ENTER-Befehl benutzt den Kanal 7. Wenn dieser schon benutzt ist, dann wird er geschlossen und bleibt auch nach Beendigung des Befehls geschlossen. Daher sollte man in Programmen, die mit ENTER arbeiten, die Benutzung von Kanal 7 vermeiden.

Die Benutzung des ENTER-Befehls mit dem Kassettenrecorder ist identisch mit der des CSAVE-Befehls. Nachdem der Kontrollton ertönt ist, muß das Band an die gewünschte Stelle gespult, PLAY auf dem Kassettenrecorder und irgendeine Taste außer BREAK auf der Computertastatur gedrückt werden.

Wird die Ausführung von ENTER unterbrochen, so bleiben alle bis dahin geladenen Programmzeilen im Speicher.

Probieren Sie doch auch einmal „ENTER“E:“ “ aus!

- Dieser Befehl entspricht dem Microsoft-BASIC-Befehl MERGE.

EOF (MSB)

Form: EOF (nausdr)
nausdr: Kanalnummer
Beispiel: IF NOT EOF(1) THEN RETURN

Mit der Funktion EOF kann man abfragen, ob auf dem angegebenen Kanal noch Daten zur Verfügung stehen, d. h. ob das Dateiende erreicht ist. Liefert die Funktion den Wert „1“, dann ist das Dateiende erreicht worden.

ERASE (BASIC XL)

Abkürzung: ER.
Form: ERASE zausdr
zausdr: Dateispezifikation
Beispiel: ERASE "D:TEST.BAS“

Der Befehl ERASE wird zum Löschen von Diskettendateien benutzt.

- Beim Löschvorgang wird daher die gesamte Datei neu eingelesen, weil das Diskettenbetriebssystem feststellen muß, welche Sektoren die Datei belegt hatte. Daraufhin werden diese Sektoren in der Belegungstabelle als frei gekennzeichnet und die Eintragung im Directory gelöscht. Die Datei selbst bleibt unberührt und kann notfalls mit einem Sektorveränderungsprogramm wie DISK-FIXER reaktiviert werden.
- Im Microsoft-BASIC wird die gleiche Aufgabe durch KILL erfüllt; im Standard-BASIC kann diese Funktion mit Hilfe eines XIO-Befehls simuliert werden.

ERL (MSB)

Form: ERL
Beispiel: PRINT "Zeilennummer: ";ERL

Mit der Funktion ERL kann abgefragt werden, in welcher Zeile der letzte Fehler aufgetreten ist.

Diese Funktion entspricht ERR(1) in BASIC XL.

ERR (BASIC XL, MSB)

Form: ERR (MSB)
ERR (nausdr) (BASIC XL)
Beispiel: FEHLER=ERR (MSB)
FEHLER=ERR(0) (BASIC XL)

Die Funktion ERR liefert in Microsoft-BASIC die Nummer des aufgetretenen Fehlers.

In BASIC XL liefert ERR(0) die Fehlernummer, ERR(1) die Nummer der Zeile, in der der Fehler aufgetreten ist.

Error

Error ist das englische Wort für Fehler. Atari-BASIC gibt bei auftretenden Fehlern Fehlernummern aus, die es ermöglichen, den Fehler schneller zu finden. Im folgenden eine Übersicht mit den Bedeutungen der Fehlernummern:

Error 2 – Memory insufficient
Der Speicherplatz reicht nicht aus. Es gibt verschiedene Ursachen: zu groß dimensionierte Datenfelder, zu stark verschachtelte Programme durch FOR-NEXT-Schleifen oder Unterprogramme oder zum Beispiel der Aufruf einer Graphikstufe, für die nicht genug Speicherplatz vorhanden ist.

Error 3 – Value Error
Es wurde ein numerischer Ausdruck angegeben, der nicht bearbeitet werden kann. Das kann daran liegen, daß er zu groß oder zu klein ist oder das Vorzeichen nicht stimmt.

Error 4 – Too Many Variables
Es wurde die Höchstzahl von 128 Variablen überschritten (über Behebung des Fehlers unter LIST nachlesen).

Error 5 – String Length Error
Es wurde versucht, einen Teil einer Zeichenkette zu lesen oder zu verändern, der nicht existiert.

Error 6 – Out of Data Error
Es wurden mehr READ-Befehle durchgeführt, als Daten hinter DATA angegeben sind. Anzahl der Daten und READ-Befehle überprüfen oder RESTORE benutzen.

Error 7 – Number greater than 32767
Ein numerischer Ausdruck der zwischen 0 und 32767 liegen müßte, hat den falschen Wert.

Error 8 – Input Statement Error
Es wurde bei einem INPUT-Befehl versucht, einen nicht zu der Variablen passenden Datentyp einzugeben (z. B. Wörter in eine numerische Variable).

Error 9 – Array or String DIM Error
Entweder wurde versucht, eine bereits dimensionierte Variable noch einmal zu dimensionieren, die angegebene Länge der Zeichenkette war größer als 32767, oder es wurde auf ein noch nicht dimensioniertes Datenfeld oder eine noch nicht dimensionierte Variable zugegriffen.

Error 10 – Argument Stack Overflow
Das angegebene Argument ist zu komplex, weil es entweder zu viele ineinander verschachtelte Variablen oder Klammern enthält.

Error 11 – Floating Point Underflow/Overflow Error
Es ist eine Zahl aufgetaucht, die außerhalb des Rechenbereichs liegt.

Error 12 – Line Not Found
Die bei einer Verzweigung, einem Sprung oder RESTORE angegebene Zeilennummer ist nicht vorhanden.

Error 13 – No Matching FOR Statement
Es wurde ein NEXT-Befehl gefunden, zu dem kein passender FOR-Befehl vorhanden ist. Möglicherweise stimmt die Verschachtelung zweier Schleifen nicht, oder es wurde ein POP-Befehl durchgeführt, ohne die FOR-NEXT-Schleife zu verlassen.

Error 14 – Line Too Long Error
Ein Befehl ging über das Ende einer logischen Zeile hinaus. Warnton am Ende vor dem Ende einer logischen Zeile beachten!

Error 15 – GOSUB or FOR-Line Deleted
Beim Rücksprung nach RETURN oder NEXT wurde die entsprechende Zeile mit GOSUB oder FOR nicht wiedergefunden.

Error 16 – RETURN Error
Das Programm ist auf RETURN getroffen, ohne vorher einen GOSUB-Befehl ausgeführt zu haben.

Error 17 – Garbage Error
Es wurde ein Befehl gefunden, der nicht ausführbar ist. Möglicherweise ist das Programm durch POKE oder USR zerstört worden.

Error 18 – Invalid String Character
Bei der Benutzung von VAL wurde eine Zeichenkette angegeben, deren erste Zeichen sich nicht in einen numerischen Ausdruck verwandeln ließen.

Error 19 – LOAD Program Too Long
Der Speicherplatz ist nicht groß genug, um das angegebene Programm zu laden.

Error 20 – Device Number Larger
Es wurde versucht einen Kanal zu benutzen, der nicht vorhanden ist (Nummer größer als 7) oder der durch den Bildschirm (Kanal 0) belegt ist.

Error 21 – LOAD File Error
Es wurde versucht eine Datei mit LOAD zu laden, die nicht mit SAVE abgespeichert worden ist.

Error 128 – BREAK Key Abort
Es wurde während eines Ein- oder Ausgabevorgangs BREAK gedrückt.

Error 129 – IOCB Already Open
Es wurde versucht, einen Kanal zu öffnen, der schon geöffnet war.

Error 130 – Nonexistent Device
In der Dateispezifikation wurde ein Gerätename angegeben, der nicht existiert.

Error 131 – IOCB Write Only
Es wurde versucht, über einen Kanal, der nur für Ausgabe geöffnet war, Daten zu lesen.

Error 132 – Invalid Command
Es wurde versucht, ein unbekanntes Ein- oder Ausgabekommando einzugeben (tritt meist mit XIO auf).

Error 133 – Device or File Not Open
Es wurde versucht, einen noch nicht geöffneten Kanal zu benutzen.

Error 134 – Bad IOCB Number
Es wurde eine Kanalnummer angegeben, die größer als 7 oder kleiner als 0 war.

Error 135 – IOCB Read Only Error
Es wurde versucht, über einen Kanal, der nur für Eingabe geöffnet war, Daten auszugeben.

Error 136 – End Of File
Beim Lesen wurde das Ende der Datei erreicht.

Error 137 – Truncated Record
Der Abstand zwischen zwei RETURN-Characters in der Datei war so groß, daß sich der Abschnitt nicht in einem Stück lesen ließ (tritt bei INPUT und ENTER auf).

Error 138 – Device Timeout
Das angegebene Gerät hat nicht innerhalb der gesetzten Frist geantwortet. Kabel, Anschlüsse, Netzschalter und ON-LINE-Schalter überprüfen!

Error 139 – Device NAK
Das Pheripherigerät konnte ein Kommando nicht ordnungsgemäß ausführen.

Error 140 – Serial Bus
Bei der Datenübertrgung über den seriellen Bus kam es zu einem Fehler.

Error 141 – Cursor Out Of Range
Die Reihen- oder Spaltenzahl der Cursorposition hat die erlaubten Grenzen über- oder unterschritten.

Error 142 – Serial Bus Data Frame Overrun
Bei der Übertragung von Daten über den seriellen Bus kam es zu einem Fehler.

Error 143 – Serial Bus Data Frame Checksum Error
Bei der Berechnung der Prüfsumme wurde ein Fehler festgestellt.

Error 144 – Device Done
Entweder konnte die Diskette nicht beschrieben werden, weil sie schreibgeschützt ist, oder es tauchte bei dem Versuch, einen bestimmten Sektor zu lesen, ein Fehler auf.

Error 145 – Read after Write Compare Error
Bei der Überprüfung des Geschriebenen wurde ein Schreibfehler festgestellt.

Error 146 – Function not implemented
Es wurde versucht, einem Peripheriegerät einen unbekannten Befehl zu geben (z. B. Formatieren der Kassette).

Error 147 – Insufficient RAM
Es ist nicht genug Speicherplatz vorhanden, um die gewünschte Graphikstufe einzuschalten.

Error 160 – Drive number error
Es wurde versucht, eine nicht existierende Diskettenstation anzusprechen.

Error 161 – Too Many Open Files
Es wurde versucht, mehr Dateien zu eröffnen, als es möglich ist.

Error 162 – Disk Full
Beim Schreiben auf Diskette wurde festgestellt, daß kein freier Sektor mehr vorhanden ist.

Error 163 – Unrecoverable Data I/O Error
Es wurde ein Fehler festgestellt, dessen Ursache unbekannt ist.

Error 164 – File number mismatch
Es wurde versucht, einen Sektor zu lesen oder zu schreiben, der nicht zu der benutzten Datei gehört.

Error 165 – File Name Error
Es wurde ein Programmname benutzt, der Graphikzeichen oder Kleinbuchstaben enthält.

Error 166 – POINT Data Length Error
Das Argument des POINT-Befehls war fehlerhaft.

Error 167 – File locked
Es wurde versucht, eine geschützte Datei zu beschreiben oder zu löschen.

Error 168 – Command Invalid
Es wurde ein Kommando benutzt, das nicht existiert (bei XIO).

Error 169 – Directory Full
Die Höchstzahl von 64 Dateien auf der Diskettenseite ist bereits erreicht worden.

Error 170 – File Not Found
Eine angegebene Datei wurde auf der Diskette nicht gefunden.

Error 171 – POINT Invalid
Der angegebene Sektor gehört nicht zu der geöffneten Datei.

Error 173 – Bad Sector at Format
Bei der Überprüfung nach dem Formatiervorgang wurden zerstörte Sektoren festgestellt.

Error 174 – Duplicate Filename
Auf der Diskette befinden sich zwei Dateien mit dem gleichen Namen (wahrscheinlich aufgrund eines Fehlers bei der Anwendung des RENAME-Kommandos).

Error 175 – Bad Load File
Bei dem Programm, das geladen werden soll, handelt es sich nicht um eine ladbare Datei.

Error 176 – Bad Format
Die angegbene Datei ist nicht im richtigen Format vorhanden (siehe Unterschiede zwischen DOS 2.0 und DOS 3.0).

Error 177 – Disk Structure Error
Fehler in der Diskettenstruktur.

Error 255 – Format Error
Beim Formatieren der Diskette ist ein Fehler aufgetreten.

Achtung: die Fehler von 173 bis 177 werden nur unter DOS 3.0 erzeugt!

ERROR (MSB)

Form: ERROR nausdr
Beispiel: ERROR 170

Mit diesem Befehl kann man eine bestimmte Fehlermeldung hervorrufen, z. B. um die Reaktion des Programms auf bestimmte Fehler zu testen.

EXP

Form: EXP(nausdr)
Beispiel: X=EXP(20)

EXP berechnet den Wert der Exponentialfunktion. EXP(X) ist also e^x, wobei e die natürliche Basis 2.71828179 ist.

```
10 REM --------------------
20 REM Demonstration zu EXP
30 REM --------------------
40 REM
50 REM Hochaufloesendste Graphik
60 GRAPHICS 8
70 REM Hintergrund schwarz/zeichnen in weiss
80 SETCOLOR 2,0,0:COLOR 1
90 REM Koordinatensystem
100 PLOT 160,0:DRAWTO 160,159
110 PLOT 0,159:DRAWTO 319,159
120 REM Funktion zeichnen
130 FOR I=0 TO 319 STEP 3
140 PLOT I,159-58*EXP((I-160)/160)
150 NEXT I
160 PRINT CHR$(125);"-1                0                1"
```

Abb. E.1: EXP – Beispielprogramm



Farben

Der Atari verfügt über 256 verschiedene Farbtöne, von denen sich allerdings von BASIC aus nur bis zu 16 verschiedene Farben gleichzeitig darstellen lassen. Beim SETCOLOR-Befehl wird die Nummer des Farbtons als zweites Argument angegeben. Im folgenden eine Tabelle der Farbwerte:

Wert	Farbton	Wert	Farbton
00	Grau	08	Blau
01	Gold	09	Hellblau
02	Orange	10	Türkis
03	Rot	11	Blaugrün
04	Rosa	12	Grün
05	Violett	13	Gelbgrün
06	Purpur	14	Orangegrün
07	Blau	15	Hellorange

```
10 REM ----------------------
20 REM Demonstration zu Farben
30 REM ----------------------
40 GRAPHICS 11:REM fuer alle Farben
50 FOR I=0 TO 191
60 COLOR I/12+0.49:IF I/12=INT(1/12) THEN COLOR 0
70 PLOT 0,I:DRAWTO 79,I
80 NEXT I
90 GOTO 90
```

Abb. F.1: Farben – Beispielprogramm

FAST (BASIC XL)

Abkürzung: FA.
Form: FAST
Beispiel: FAST

Mit dem Befehl FAST kann man die Ausführung des nachfolgenden Programms beschleunigen. (Für Fortgeschrittene: Die Zeilennummern werden für die Dauer der Programmausführung durch die tatsächlichen Adressen der Programmzeilen im Speicher ersetzt, so daß die Suchzeit nach der nächsten Zeile wegfällt; daher tritt auch kein Geschwindigkeitsgewinn ein, wenn man als Zeilennummern keine Konstanten verwendet.)

Anmerkungen

- Der FAST-Modus wird automatisch eingeschaltet, wenn man ein Programm mit „RUN Dateinamen" startet und dabei nicht „SELECT" gedrückt wird.

Fehler

Fehlermeldungen

Eine Fehlermeldung wird ausgegeben, wenn bei der Abarbeitung irgendeines Befehls ein Fehler aufgetreten ist. Während Microsoft-BASIC II nur ungenaue Fehlerbeschreibungen und BASIC XL zu jedem Fehler einen Text ausgibt, gibt Atari-BASIC nur die Nummer des Fehlers aus.

- Ein Programm zur Ausgabe von deutschen Fehlermeldungen steht als Beispielprogramm auf S. 79.
- Zum Abfangen von Fehlern gibt es in Atari-BASIC und BASIC XL den Befehl TRAP, in Microsoft-BASIC den Befehl ON ERROR. Mit Hilfe dieser Befehle kann man veranlassen, daß das Programm im Falle eines Fehlers an einer vom Programmierer bestimmten Programmzeile fortgesetzt wird.

```
100 REM ------------------------
110 REM Deutsche Fehlermeldungen
120 REM ------------------------
130 REM Zeichenkette dimensionieren
140 DIM FEHLER$(40)
150 REM Fehler abfangen
160 TRAP 160
170 NUMMER=195
180 ZEILE=186
190 ANZAHL=53:REM Anzahl der moegliche
n Fehler
200 RESTORE 220
210 FOR I=1 TO ANZAHL
220 REAREAD FNUMMER,FEHLER$
230 IF FNUMMER=PEEK(NUMMER) THEN PRINT
 CHR$(253);FEHLER$;", Zeile ";PEEK(ZEI
LE)+256*PEEK(ZEILE+1):GOTO 780
240 NEXT I
250 DATA 2,Zu wenig Speicherplatz
260 DATA 3,Falscher Wert
270 DATA 4,Zu viele Variablen
280 DATA 5,Laenge der Zeichenkette
290 DATA 6,Keine Daten mehr
300 DATA 7,Zahl >32767
310 DATA 8,Eingabefehler
320 DATA 9,Dimensionierungsfehler
330 DATA 10,Zu komplexer Term
340 DATA 11,Zahlenbereich verlassen
350 DATA 12,Zeile nicht gefunden
360 DATA 13,Kein passender FOR-Befehl
370 DATA 14,Zeile zu lang
380 DATA 15,GOSUB- oder FOR-Zeile gelo
escht
390 DATA 16,Fehlendes GOSUB
400 DATA 17,Falscher Befehl
410 DATA 18,Keine Zahl bei VAL
420 DATA 19,Programm zu lang
430 DATA 20,Kanalnummer ungueltig
440 DATA 21,Keine SAVE-Datei
450 DATA 128,BREAK gedrueckt
460 DATA 129,Kanal schon offen
470 DATA 130,Geraet nicht vorhanden
480 DATA 131,Nur fuer Ausgabe offen
490 DATA 132,Ungueltiger Befehl
500 DATA 133,Kanal nicht offen
510 DATA 134,Falsche Kanalnummer
520 DATA 135,Nur fuer Eingabe offen
530 DATA 136,Ende der Datei
540 DATA 137,Record zu lang
550 DATA 138,Geraet antwortet nicht
560 DATA 139,Geraet arbeitet fehlerhaf
t
570 DATA 140,Fehler am seriellen Bus
580 DATA 141,Cursor ausserhalb der Gre
nzen
590 DATA 142,Uebertragung am seriellen
 Bus fehlerhaft
600 DATA 143,Pruefsummenfehler
610 DATA 144,Sektor kaputt oder schrei
bgeschuetzt
```

Abb. F.2: Fehler – Beispielprogramm

```
620 DATA 145,Fehler bei Ueberpruefung
630 DATA 146,Kommando hier nicht ausfu
ehrbar
640 DATA 147,Zu wenig Platz fuer GRAPH
ICS
650 DATA 160,Diskettenstation nicht vo
rhanden
660 DATA 161,Zu viel Dateien offen
670 DATA 162,Diskette voll
680 DATA 163,Unbekannter Fehler!
690 DATA 164,Sektor gehoert nicht zur
Datei
700 DATA 165,Falscher Dateiname
710 DATA 166,POINT-Befehl fehlerhaft
720 DATA 167,Datei geschuetzt
730 DATA 168,Ungueltiger Befehl
740 DATA 169,Directory voll
750 DATA 170,Datei nicht gefunden
760 DATA 171,POINT-Befehl ungueltig
770 DATA 255,Formatierfehler
780 POKE 186,0:POKE 187,0:POKE 195,0
```

Abb. F.2: Fehler – Beispielprogramm (Forts.)

```
?1/0
Zahlenbereich verlassen, Zeile 0

READY
RETURN
Fehlendes GOSUB, Zeile 0

READY
?VAL("QQ")
Keine Zahl bei VAL, Zeile 0

READY
DIM A(9999)
Dimensionierungsfehler, Zeile 0

READY
LIST 60000

Zahl >32767, Zeile 0
```

Abb. F.3: Fehler – Beispiellauf

Feld

Ein Feld ist eine Datenstruktur, die vor allem zur Abspeicherung von Tabellen und Datensätzen benutzt wird. Es wird durch den Befehl DIM initialisiert, wobei der Name der Feldvariablen sowie die Anzahl und die Größe der Dimensionen angegeben wird.

FILL (MSB)

Form: FILL nausdr1,nausdr2 TO nausdr3,nausdr4
nausdr1/2: Koordinaten des Anfangspunktes der Grundlinie
nausdr3/4: Koordinaten des Endpunktes der Grundlinie
Beispiel: FILL X,Y TO A,B

Der Befehl FILL füllt einen bestimmten Teil des Bildschirms mit der laufenden Zeichenfarbe auf. Dabei müssen zunächst drei Seiten der Fläche mit PLOT...TO gemalt werden, bevor der Füllvorgang durch Angabe von Anfangs- und Endpunkt der letzten Linie hinter FILL gestartet wird.

In den anderen BASIC-Dialekten kann man die FILL-Funktion mit Hilfe eines XIO-Befehls simulieren.

FIND (BASIC XL)

Form: FIND(zausdr1,zausdr2,nausdr)
zausdr1: Zeichenkette, in der gesucht werden soll
zausdr2: Zeichenkette, die gesucht werden soll
nausdr: Nummer des Zeichens, bei dem die Suche begonnen werden soll
Beispiel: FIND (DAT$,SUCH$,45)

Die Funktion FIND sucht die zweite angegebene Zeichenkette in der ersten. Das dritte Argument gibt die Nummer des Zeichens an, von dem ab in der ersten Zeichenkette gesucht werden soll. So ist es auch möglich, mehrfach auftretende Teile zu finden.

Im Microsoft-BASIC heißt der entsprechende Befehl INSTR.

Formatieren

Jede Diskette muß, bevor sie mit Daten beschrieben werden kann, formatiert werden. Dabei wird sie in Sektoren unterteilt; Systemsektoren (Bootsektoren), Belegungstabelle (VTOC) und Dirctory werden geschrieben.

- Der Formatiervorgang kann entweder durch einen besonderen XIO-Befehl (→ XIO) oder den entsprechenden DOS-Befehl ausgelöst werden.

FOR...TO...STEP...NEXT

Abkürzung: F./N.
Form: FOR numvar=nausdr TO nausdr STEP nausdr
NEXT numvar
Beispiel: F. I=1 TO 16
N. I

Einfache Programmschleifen: Mit dem Befehl FOR kann man erreichen, daß ein bestimmter Programmteil mehrfach wiederholt wird. Dazu gibt man hinter FOR den Namen der Schleifen- oder Indexvariablen an. Diese Variable verändert bei jedem Durchlauf der Programmschleife ihren Wert (mit einer Ausnahme). Die Abarbeitung der Schleife wird beendet, wenn die Kontrollvariable einen bestimmten Wert erreicht hat. Der Anfangswert der Kontrollvariablen wird hinter dem Gleichheitszeichen, der Endwert hinter TO angegeben.

Bei jedem Durchlauf der Programmschleife wird die Kontrollvariable um 1 erhöht, bis sie den Endwert erreicht hat.

Das Ende der Schleife wird durch den Befehl NEXT gekennzeichnet. Erreicht das Programm diesen Befehl, dann wird die Kontrollvariable erhöht und überprüft, ob sie den Endwert überschritten hat. Ist dies geschehen, dann wird das Programm bei der Anweisung hinter FOR, sonst hinter NEXT fortgesetzt.

```
100 FOR I=32 TO 91
110 PRINT CHR$(I);
120 NEXT I
```

Dieses Programm würde also zunächst den Wert von I auf 32 setzen, das entsprechende ASCII-Zeichen ausgeben, den Wert von I um eins erhöhen und dann überprüfen, ob I 96 überschritten hat. Wenn nicht, dann wird das Programm mit Zeile 110 fortgesetzt. Insgesamt wird das Programm am Ende alle ATASCII-Zeichen vom Leerzeichen bis zum Z ausgegeben haben.

Schleifen mit veränderter Schrittweite

Natürlich muß die Kontrollvariable nicht um den Wert 1 verändert werden. Mit STEP ist es möglich, zu bestimmen, um welchen Wert die Kontrollvariable erhöht werden soll. Dabei sind auch negative Werte (die Kontrollvariable wird dann verringert) und sogar der Wert Null möglich (die Schleife wird endlos fortgesetzt, da sich die Kontrollvariable nicht verändert). Selbstverständlich wird es mit großer Wahrscheinlichkeit zu Fehlern kommen, wenn der Wert der Kontrollvariablen innerhalb der Schleife geändert wird.

Verschachtelte Schleifen

Außerdem ist es möglich, Programmschleifen ineinander zu verschachteln. Dabei müssen die NEXT-Befehle in der umgekehrten Reihenfolge der FOR-Befehle stehen, so daß es eine innere und eine äußere Schleife gibt.

```
100 FOR FARBE=0 TO 15
110   FOR HELLIGKEIT=0 TO 14 STEP 2
120     SETCOLOR 2,FARBE,HELLIGKEIT
130   NEXT HELLIGKEIT
140 NEXT FARBE
```

Dieses Programm würde also nacheinander für alle Farbtöne alle Helligkeiten einschalten. Zur Hervorhebung der Programmstruktur haben wir hier die Schleifen eingerückt (das macht das BASIC XL übrigens automatisch). Selbstverständlich ist es auch möglich, mehr als zwei Schleifen ineinander zu verschachteln.

Vorzeitiger Abbruch

Um bei Auftreten des Befehls NEXT den entsprechenden FOR-Befehl wiederfinden zu können, muß das BASIC die betreffende

Zeilennummer zwischenspeichern. Ist die Programmschleife beendet, wird dieser Speicher wieder gelöscht. Wenn man nun aber die Schleife verläßt, ohne den letzten NEXT-Befehl auszuführen, kann es zu Fehlern kommen. Daher gibt es den Befehl POP, durch den man diese interne Zwischenspeicherung beenden kann (siehe Demoprogramm).

```
10 REM -------------------------
20 REM Demonstration zu FOR/NEXT
30 REM -------------------------
40 REM Das kleine Einmaleins
50 GRAPHICS 0
60 FOR X=0 TO 9
70 REM aeussere Schleife
80 FOR Y=0 TO 9
90 REM innere Schleife
100 REM Wird 10*10=100 mal durchlaufen

110 POSITION X*3+3,Y*2+3
120 IF X*Y<10 THEN PRINT " ";
130 PRINT X*Y
140 NEXT Y
150 NEXT X
```

Abb. F.4: FOR...NEXT – Beispielprogramm

```
0  0  0  0  0  0  0  0  0  0

0  1  2  3  4  5  6  7  8  9

0  2  4  6  8 10 12 14 16 18

0  3  6  9 12 15 18 21 24 27

0  4  8 12 16 20 24 28 32 36

0  5 10 15 20 25 30 35 40 45

0  6 12 18 24 30 36 42 48 54

0  7 14 21 28 35 42 49 56 63

0  8 16 24 32 40 48 56 64 72

0  9 18 27 36 45 54 63 72 81
```

Abb. F.5: FOR...NEXT – Beispiellauf

Ein sehr häufiger Programmierfehler ist, den NEXT-Befehl hinter IF zu setzen.

```
100 FOR I=1 TO LEN(A$)
110 IF A$(I,I)=CHR$(0) THEN A$(I,I)=" ":NEXT I
```

Das obenstehende Beispielprogramm würde seine Aufgabe, innerhalb der Zeichenkette alle Zeichen mit dem Wert 0 durch ein Leerzeichen zu ersetzen, höchstwahrscheinlich nicht erfüllen, da der Befehl NEXT nicht gefunden wird, wenn die Bedingung bei IF nicht wahr ist.

FRE

Form: FRE(0)
Beispiel: FREI=FRE(0)

Die Funktion gibt die Größe des noch freien Speicherplatzes aus. Dabei ist das eine Argument der Funktion ohne Wirkung (also eine sogenannte DUMMY-Variable).

- Mit Hilfe der FRE-Funktion kann man auch feststellen, wieviel Speicherplatz eine bestimmte Dimensionierung verbraucht, indem man einfach vorher FRE(0) in einer Variablen ablegt und den neuen mit dem alten Wert vergleicht.

Funktion

Eine Funktion ist ein eingebautes Unterprogramm, das meistens bestimmte Berechnungen durchführt. Sie wird einfach durch die Nennung ihres Namens aufgerufen. Dem Funktionsnamen muß mindestens ein Argument (numerischer oder Zeichenkettenausdruck) in Klammern folgen (→ Argument). Der Funktionsaufruf wird dann bei der Auswertung des Ausdrucks durch den Wert ersetzt, den die Funktion berechnet.

G

Gerätenamen

Bei der Angabe einer Dateispezifikation muß stets zuerst der Gerätename angegeben werden. Im Normalzustand kennt der Atari folgende Gerätenamen:

C: – Cassette (Kassettenrecorder)
E: – Editor (Textbildschirm)
K: – Keyboard (Tastatur)
P: – Printer (Drucker)
S: – Screen (Graphikbildschirm)

Weitere reservierte Gerätenamen sind:

D: – Diskettenstation (wird durch das Diskettenbetriebssystem definiert)
R: – RS-232-Schnittstelle (wird durch ein besonderes Programm eingeschaltet)

Da es in Maschinensprache nicht schwer ist, eigene „Gerätenamen" zu definieren, möchte ich hier noch eine kurze Liste von Gerätenamen geben, die bereits durch spezielle Programme erzeugt worden sind. Das könnte helfen, Überschneidungen mit eigenen Programmen zu vermeiden.

G: – 3D-Graphik-Hilfsprogramm
M: – Benutzung des RAMs als Speicher für Dateien
T: und V: – Programm zur Ausgabe von Text in anderen Graphikstufen als GRAPHICS 0.
V: – „VOICE" – Zur Ansteuerung der Atari-Sprachbox

GET

Abkürzung: GE.
Form: GET#nausdr,varname
nausdr: Kanalnummer
varname: Variable, in das gelesene Byte übertragen wird
Beispiel: GET#3,TASTE

Mit dem Befehl GET kann von einem beliebigen Peripheriegerät ein einzelnes Byte in die angegebene Variable eingelesen werden. Dabei wird der Eingabekanal verwendet, der hinter dem Zeichen „#“ angegeben ist.

GET von der Tastatur (“K:“ (Keyboard))
Wird mit GET von der Tastatur ein Byte eingelesen, dann wird zunächst überprüft, ob im Tastaturspeicher noch ein nicht verarbeiteter Tastendruck vorliegt (der Atari merkt sich stets das zuletzt eingetippte Zeichen). Wenn nicht, dann wartet er darauf, daß der Benutzer eine Taste drückt und liefert dann den ATASCII-Wert (→ ATASCII) dieses Zeichens zurück.

GET vom Bildschirm („E:“ (Editor) oder „S:“ (Screen))
Die Benutzung des GET-Befehls mit dem Bildschirm ist ein wenig problematisch. Verwendet man den Gerätenamen „E:“, dann wird nicht ein Zeichen eingelesen, sondern wie bei dem Befehl INPUT verfahren und dann der ATASCII-Wert des ersten Zeichens der eingegebenen Zeichenkette zurückgegeben. Bei der Verwendung des Namens „S:“ ist es theoretisch möglich, festzustellen, welches Zeichen an der momentanen Bildschirmposition steht. Dafür gibt es jedoch den besonderen Befehl LOCATE, mit dem man problemlos jede Bildschirmposition überprüfen kann.

GET von der Diskette (“Dn:name.erweiterung“)
Bei GET von der Diskettenstation wird das nächste Byte der geöffneten Diskettendatei übertragen. Da Bytes auf der Diskette blockweise abgespeichert sind und das Diskettenbetriebssystem diese Blöcke in einem Stück liest und abspeichert, wird das Diskettenlaufwerk nur ungefähr alle 125 Zeichen zu laufen beginnen.

GET vom Kassettenrecorder ("C:")
Die Verwendung des Befehls GET im Zusammenhang mit dem Kassettenrecorder läuft ähnlich ab wie mit der Diskettenstation. Auch hier wird immer ein ganzer Block in den Speicher eingelesen oder abgespeichert, woraufhin der Motor des Kassettenrecorders stoppt.

GOSUB

Abkürzung: GOS.
Form: GOSUB nausdr
nausdr: Zeilennummer
Beispiel: GOS. 32000

Der Befehl GOSUB übergibt die Kontrolle an ein Unterprogramm an der angegebenen Zeilennummer. Dabei merkt sich der Rechner, von welcher Stelle im Programm das Unterprogramm aufgerufen worden ist. Wenn am Ende des Unterprogramms der Befehl RETURN auftaucht, wird das Programm bei dem Befehl fortgesetzt, der hinter dem GOSUB-Befehl steht, der das Unterprogramm aufgerufen hat.

Als Argument kann auch statt einer Konstanten ein numerischer Ausdruck angegeben werden, aus dem dann die Zeilennummer berechnet wird. Durch „GOSUB 1000+AUSWAHl*100" könnte man also verschiedene Unterprogramme bei den Zeilennummern 1000,1100 etc. von dem Wert der Variablen „AUSWAHL" (0,1,2,...) abhängig aufrufen. Allerdings sollte man daran denken, daß kein BASIC-Compiler dazu fähig ist, solche Ausdrücke zu verarbeiten, und daß es auch bei der Benutzung eines Programms (oder Befehls) zur Neunumerierung der Zeilennummern zu Fehlern kommen wird.

Gibt es keine Programmzeile mit der angegebenen oder errechneten Zeilennummer, dann kommt es zu einem Fehler.

Will man unbedingt ein Unterprogramm vor seinem Ende verlassen, ohne den RETURN-Befehl auszuführen, muß man vorher einen POP-Befehl durchführen, damit der Rechner die Zeilennummer des letzten GOSUB, die er sich gemerkt hat, vergißt (→ FOR, POP).

Anmerkungen

- Unterprogramme sollte man immer dann benutzen, wenn ein bestimmter Programmteil mehrmals an verschiedenen Stellen im Programm auftaucht. Das Unterprogramm darf an jeder Stelle im Speicher beginnen, aufgrund der Arbeitsweise von BASIC wird das Programm aber dann schneller ablaufen, wenn die Unterprogramme so weit wie möglich am Anfang des Programms stehen.

```
10 REM ------------------------
20 REM Demonstration zu GOSUB
30 REM ------------------------
40 REM
50 GRAPHICS 8+16:SETCOLOR 2,0,0:COLOR 1
60 DIM TEXT$(255)
70 DEG
80 PLOT 0,96:DRAWTO 319,96
90 PLOT 160,0:DRAWTO 160,191
100 REM Sinuskurve
110 FOR I=0 TO 319 STEP 3
120 PLOT I,96+85*SIN(1*9/8)
130 NEXT I
140 REM Graph beschriften
150 REM Position in X,Y
160 REM Text in Text$
170 X=0:Y=0:TEXT$="Sinuskurve"
180 GOSUB 1000
190 X=21:Y=100:TEXT$="0":GOSUB 1000
200 X=21:Y=0:TEXT$="1":GOSUB 1000
210 X=21:Y=183:TEXT$="-1":GOSUB 1000
250 GOTO 250
999 REM Unterprogramm zur Ausgabe von Zeichen in Graphics 8
1000 ZEICHENSATZ=PEEK(756)*256:REM Anfangsadresse des Zeichensatzes
1005 SCR=PEEK(88)+256*PEEK(89):REM Anfangsadresse des Bildspeichers
1010 FOR I=1 TO LEN(TEXT$)
1020 PNT=SCR+X+40*Y
1030 WERT=ASC(TEXT$(I)):INV=0:IF WERT>127 THEN INV=1:WERT=WERT-128
1040 IF WERT<32 THEN WERT=WERT+64:GOTO 1060
1050 IF WERT<96 THEN WERT=WERT-32
1060 PNT2=ZEICHENSATZ+8*WERT
1070 FOR J=0 TO 7
1075 INHALT=PEEK(PNT2+J):IF INV THEN INHALT=255-INHALT:REM Invers
1080 POKE PNT+J*40,INHALT
1090 NEXT J
1100 REM 'Cursor' weiterbewegen
1110 X=X+1:IF X=40 THEN X=0:Y=Y+8
1120 NEXT I
1130 RETURN
```

Abb. G.1: GOSUB – Beispielprogramm

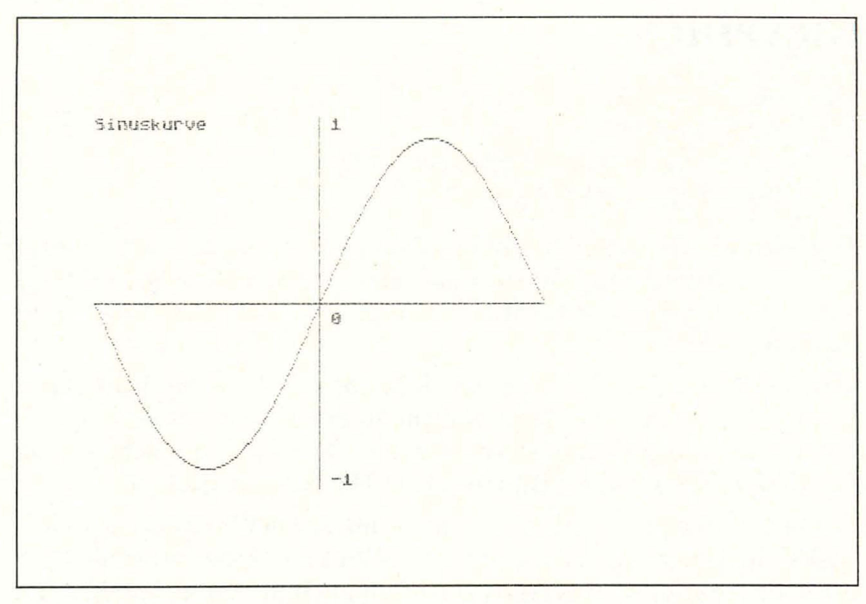


Abb. G.2: GOSUB – Beispiellauf

GOTO/GO TO

Abkürzung: G.
Form: GOTO nausdr
GO TO nausdr
nausdr: Zeilennummer
Beispiel: GOTO 10
GO TO 1999

Das Argument des GOTO-Befehls kann genau wie bei GOSUB eine Konstante oder ein numerischer Ausdruck sein. Das Programm wird mit der ersten Anweisung der angegebenen Zeile fortgesetzt. Existiert die angegebene Zeile nicht, dann wird ein Fehler auftreten.

Befehle, die hinter GOTO in einer Programmzeile stehen, werden nie ausgeführt werden, da die Abarbeitung einer Programmzeile stets nach GOTO aufhört.

GRAPHICS

Abkürzung: GR.
Form: GRAPHICS nausdr
nausdr: Nummer der Graphikstufe
Beispiel: GRAPHICS 8+16

Der Befehl GRAPHICS schaltet die durch das angegebene Argument spezifizierte Graphikstufe an. Außerdem werden die Farbregister auf die Standardwerte zurückgesetzt und der Bildschirm gelöscht.

GRAPHICS ohne Löschen des Bildschirms: Soll der Bildschirm nicht gelöscht werden, dann braucht zu der gewünschten Graphikstufe nur eine 32 addiert zu werden. So ist es möglich, ein in GRAPHICS 8 gemaltes Bild in GRAPHICS 14 anzusehen.

Graphikstufen mit Textfenster addiert man zu der Nummer der Graphikstufe 16, dann werden die untersten vier Bildschirmzeilen für normale Textzeilen reserviert (nicht möglich in den Graphikstufen 9–11). Obwohl es bei Eingabe von GRAPHICS 16 zu einem Fehler kommt, ist es möglich, eine Graphikstufe mit 20+4 Textzeilen zu erzeugen, wobei die untersten vier Zeilen wie ein normales Textfenster und der obere Teil genauso wie in den Graphikstufen 1 oder 2 angesprochen werden. Eine solche Bildschirmaufstellung ist immer dann praktisch, wenn eine Eingabe erfolgen soll, ohne daß andere Teile des Bildschirms zerstört werden können. Um also auch im normalen Textmodus ein Textfenster erzeugen zu können, bedient man sich einfach des Registers 703, das die Anzahl der normalen Textzeilen auf dem Bildschirm festlegt (leider sind nur die Werte 0, 4 und 24 erlaubt). Durch „POKE 703,4“ wird das Textfenster aktiviert, mit „POKE 703,24“ wieder abgeschaltet.

Wenn kein Textfenster vorhanden ist, können weder Texte mit PRINT auf den Bildschirm geschrieben noch über INPUT Werte vom Bildschirm eingelesen werden. Daher wird auch jede Graphikstufe ohne Textfenster sofort dann gelöscht, wenn eine Fehlermeldung ausgegeben werden muß, oder wenn das Programm beendet ist. Daher muß man, wenn man am Ende eines Programms die Rückkehr in den Textmodus nicht wünscht, das Programm mit einer endlosen Schleife beenden, die dann mit BREAK unterbrochen werden kann.

Die verschiedenen Graphikstufen

Nummer	Art	Auflösung	Farben	Speicherbedarf
0	Text	40* 24	2	
1	Text	20* 24	5	
2	Text	20* 12	5	
3	Punkt	40* 24	4	
4	Punkt	48* 80	2	
5	Punkt	48* 80	4	
6	Punkt	96*160	2	
7	Punkt	96*160	4	
8	Punkt	192*320	2	
9	Punkt	192* 80	16	
10	Punkt	192* 80	9	
11	Punkt	192* 80	16	
12	Text	40* 24	5	
13	Text	40* 12	5	
14	Punkt	192*160	2	
15	Punkt	192*160	4	

- Für die Graphikstufe 0 muß folgendes beachtet werden: Wenn man auf die oben beschriebene Art und Weise ein Textfenster eingeschaltet hat, kann man die oberen zwanzig Bildschirmzeilen nur noch über Kanal 6, der für die Ausgabe auf den Bildschirm geöffnet ist, ansteuern. Da dieser Kanal aber normalerweise im Textmodus nicht geöffnet ist, muß man statt „GRAPHICS 0" „OPEN #6,12,0,"S:"" verwenden.

```
10 REM --------------------------
20 REM Graphics 0 mit Textfenster
30 REM --------------------------
40 OPEN #6,12,0,"S:":REM Kanal 6 fuer
Bildschirmausgabe initialisieren
50 REM Listing ausgeben
60 LIST
70 REM Textfenster einschalten
80 POKE 703,4:PRINT "K":REM Bild loesc
hen
90 REM Listing ausgeben
100 LIST
```

Abb. G.3: GRAPHICS 0 – Beispielprogramm

- Auf den neuen Atari-Computern ist es möglich, einen Graphikmodus einzuschalten, in dem die Buchstaben nicht buchstabenweise, sondern fließend (zeilenweise) nach oben verschoben werden („FINE-SCROLLING"). Dieser Modus kann dadurch eingeschaltet werden, daß man vor „GRAPHICS 0" „POKE 622,255" durchführt.

Anmerkungen

- In den Graphikstufen 1 und 2 hat man fünf Farben zur Auswahl: die Hintergrundfarbe und vier Zeichenfarben, die man über den ATASCII-Code (→ATASCII, COLOR) des Zeichens auswählt.
- Obwohl die Graphikstufen 4 und 6 von der Anzahl der Farben und der Auflösung her uninteressant zu sein scheinen, haben auch diese ihre Daseinsberechtigung. Für den fortgeschrittenen Programmierer dürfte es interessant und wichtig sein, daß diese beiden Graphikstufen die geringste Rechenzeit des Computers verbrauchen. Dies ist natürlich einerseits bei solchen Programmen wichtig, die möglichst schnell ablaufen sollen, aber auch dann, wenn es darauf ankommt, daß das Hauptprogramm möglichst kontinuierlich abgearbeitet wird. Dies ist beispielsweise dann der Fall, wenn man Sprachausgabe ohne Hardwarezusatz erzeugen will.
- Die Graphikstufen 9,10 und 11 unterscheiden sich wesentlich von allen anderen Graphikstufen, was man schon daran erkennen

```
30000 CLOSE #1:OPEN #1,8,0,"P:"
30010 ? #1;"␛@":? #1;"␛3";CHR$(24)
30020 SC=PEEK(88)+256*PEEK(89)
30030 FOR I=0 TO 39
30040 ? #1;"␛L";CHR$(128);CHR$(1);
30050 FOR J=7640 TO 0 STEP -40
30060 WE=PEEK(SC+J+I):PUT #1,WE:PUT #1,W
E
30070 NEXT J:? #1
30080 NEXT I:? #1:? #1;"␛@":CLOSE #1
30090 RETURN
```

Abb. G.4: GRAPHICS 8 – Beispielprogramm

kann, daß es in diesen Graphikstufen nicht möglich ist, Textfenster zu benutzen. Dies liegt daran, daß diese Graphikstufen nicht vom normalen Graphikbaustein ANTIC, sondern vom GTIA erzeugt werden, der wiederum keinen Text erzeugen kann. Eine Mischung der Graphikstufen 9 bis 11 mit anderen ist nur unter Verwendung von Maschinensprachroutinen zu erreichen.



Hardcopy

Unter einer HARDCOPY versteht man die Übertragung des Bildschirminhalts auf den Drucker. Eine Hardcopy einer Graphik läßt sich nur mit einem graphikfähigen Drucker ausgeben. Zusätzlich benötigt man ein Programm, das den Inhalt des Bildschirmspeichers in Zusammenhang mit der eingeschalteten Graphikstufe interpretiert und ihn in für den Drucker verständliche Zeichen übersetzt..

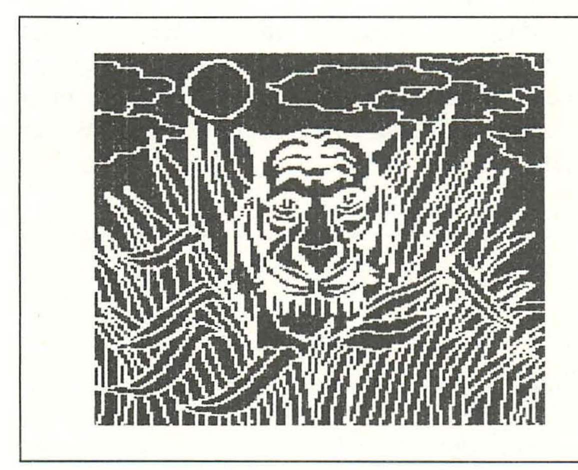


Abb. H.1: HARDCOPY – Beispiellauf

Hardware

Unter der Hardware eines Computersystems versteht man im weiteren Sinne sämtliche materiellen Bestandteile wie Tastatur, Speicher, Bildschirm oder Diskettenlaufwerk.

HELP-Taste

Die HELP-Taste ist eine neue, zusätzliche Funktionstaste auf der Tastatur des Atari 600 XL und 800 XL. Sie soll dazu dienen, dem Benutzer eines Programms in bestimmten Situationen eine praktische Hilfestellung zu geben. Da es diese Taste nicht in den alten Modellen 400/800 gegeben hat, nutzen bisher nur wenige Programme diese sinnvolle Einrichtung (Atari DOS 3.0, Atari Artist).

Für den Status der HELP-Taste besitzt der Rechner eine eigene Speicherzelle. Ist dieses Byte (732) nicht Null, dann ist die HELP-Taste gedrückt worden. Da dieses Register nicht automatisch nach Loslassen der Taste gelöscht wird, sollte man es, nachdem man es abgefragt hat, durch „POKE 732,0“ zurücksetzen.

Anmerkungen

- Wer zusätzliche Eigenschaften der XL-Modelle nutzt, sollte immer daran denken, daß viele unter uns noch einen der alten Atari-Computer besitzen. Daher ist es sinnvoll, Programme so zu schreiben, daß sie auch auf dem Atari 400 und 800 laufen. Hier könnte man beispielsweise zusätzlich eine andere Taste, z. B. das Fragezeichen, abfragen.

HEX$ (BASIC XL)

Form: HEX$(nausdr)
Beispiel: W$=HEX$(22145)

Die Funktion HEX$ rechnet die übergebene Dezimalzahl in eine hexadezimale Zahl um. Da der Rechner nicht in diesem Zahlensystem rechnen kann, ist das Ergebnis eine Zeichenkette.

Hexadezimal

Im Hexadezimalsystem gibt es statt zehn verschiedener Ziffern derer sechzehn, wobei man sich für die sechs zusätzlichen Ziffern der Ein-

fachheit halber der Buchstaben „A“ bis „F“ bedient. Der Vorteil von hexadezimalen Zahlen ist, daß man stets mit zwei Ziffern ein Byte ausdrücken kann (16*16=256), und daß sie wesentlich einfacher in das binäre Zahlensystem umzurechnen sind.

- Um eine Zahl als HEX-Zahl zu kennzeichnen, stellt man ihr gewöhnlich ein Dollarzeichen voran. Eine andere gebräuchliche Schreibweise setzt hinter die Zahl den Buchstaben „H“ für hexadezimal.

HSTICK (BASIC XL)

Form: HSTICK(nausdr)
nausdr: Nummer des Joysticks
Beispiel: X=X+HSTICK(0)

Die Funktion HSTICK erleichtert die Abfrage des Joysticks. Als Argument muß die Nummer des Joysticks angegeben werden. HSTICK liefert den Wert −1, wenn der Joystick nach links gedrückt ist und +1, wenn er nach rechts bewegt wird. Dabei werden auch diagonale Bewegungen richtig ausgewertet.

Durch diese sinnvolle Funktion kann man zum Beispiel den Ausdruck „x=x+HSTICK(0)“ verwenden, um irgendeine horizontale Koordinate je nach Joystickstellung zu verändern.

I

I/O

I/O ist die Abkürzung für INPUT/OUTPUT, zu deutsch Eingabe/ Ausgabe. Als Ein- und Ausgabegeräte bezeichnet man nicht nur Speichermedien wie das Diskettenlaufwerk, sondern auch Bestandteile des eigentlichen Computers, wie z. B. die Tastatur.

IF...THEN

Abkürzung: nicht möglich
Form: IF logausdr THEN befehl(:befehl...)
IF logausdr THEN nausdr
Beispiel: IF A=B THEN A=A+B
IF A<B THEN 2000

Befehlsstruktur
Dieser Befehl dient dazu, das Programm unter verschiedenen Situationen unterschiedlich reagieren zu lassen. Der hinter IF stehende logische Ausdruck wird zunächst ausgewertet. Ist er wahr, dann werden die hinter THEN stehenden Befehle ausgeführt. (Soll nach THEN ein Sprung (GOTO) ausgeführt werden, dann darf das Wort GOTO weggelassen werden.) Ist der Ausdruck logisch falsch, dann wird die Programmausführung bei der folgenden Programmzeile fortgesetzt.

Der logische Ausdruck
Der logische Ausdruck hinter IF kann in verschiedener Art und Weise angegeben werden. Meistens wird man wohl zwei numerische

Ausdrücke, verknüpft durch einen logischen Operator, angeben. In BASIC gibt es die folgenden logischen Operatoren:

= ist gleich
<> ist ungleich
< ist kleiner als
> ist größer als
>= ist größer oder gleich als
<= ist kleiner oder gleich als

Ein logischer Ausdruck kann auch aus mehreren Bestandteilen bestehen, die durch AND und OR untereinander in Beziehung gesetzt werden. Der Ausdruck NOT kehrt den Wahrheitswert des folgenden logischen Ausdrucks um. Er wird eigentlich nicht benötigt, da alle denkbaren logischen Operatoren existieren. Dennoch hat er einen Sinn: Durch NOT können viele logische Ausdrücke übersichtlicher und verständlicher geschrieben werden. Im folgenden einige Beispiele für logische Ausdrücke:

IF STICK(0)=14 AND NOT STRIG(0) THEN ...
„Wenn Joystick 0 nach oben gedrückt ist und der Feuerknopf berührt wird ..."
IF A>0 AND A<256 THEN ...
„Wenn A zwischen 0 und 256 liegt ..."
IF ABS(R)<2000
„Wenn der Betrag von R kleiner als 2000 ist"

Außerdem ist es auch möglich, Zeichenkettenvariablen logisch zu verknüpfen. So würde der Ausdruck „"MAIER"<"MEIER"" das Ergebnis „wahr" liefern, da der erste Name in der Reihenfolge der ATASCII-Werte (und auch in der des Alphabets) vor dem zweiten Namen steht.

Um die dritte mögliche Art eines logischen Ausdrucks zu verstehen, muß man wissen, daß der logische Wert „wahr" intern durch einen Wert ungleich Null und der logische Wert „unwahr" durch die Zahl Null repräsentiert wird. Daher ist es auch möglich, einen einfachen numerischen Ausdruck als logischen Ausdruck zu verwenden. Schreibt man beispielsweise IF STRIG(0) THEN PRINT „Knopf nicht gedrückt", dann wird der PRINT-Befehl dann ausgeführt, wenn STRIG(0) nicht den Wert Null hat.

IF...ELSE...ENDIF (BASIC XL)

```
Form: IF logausdr:Befehl(:Befehl...)
      ELSE Befehl(:Befehl...)
      ENDIF
Beispiel: 10 IF A$="A": A$="B"
          20 ELSE: A$="C"
          30 ENDIF
```

Diese erweiterte Form von IF...THEN in BASIC XL bietet zwei Vorteile gegenüber dem Standardbefehl:

– Durch die Einführung von ENDIF ist es möglich, IF-Strukturen auch über mehrere Programmzeilen gehen zu lassen. Wenn die Bedingung nicht erfüllt ist, wird das Programm bei dem auf ENDIF folgenden Befehl fortgesetzt.

– Durch ELSE kann man auch für den Fall, daß die angegebene Bedingung nicht erfüllt ist, bestimmte Befehle vorsehen.

- In der Zeile, in der der Befehl IF steht, muß mindestens noch ein weiterer Befehl folgen.

IF...THEN...ELSE (MSB)

```
Form: IF logausdr THEN Befehl ELSE Befehl
Beispiel: IF VER=ZAHL THEN PRINT "RICHTIG!" ELSE PRINT
          "FALSCH!"
```

Die erweiterte Struktur von IF...THEN...ELSE erlaubt es in Microsoft-BASIC, Befehle für den Fall anzugeben, daß der logische Ausdruck unwahr ist. In diesem Fall werden die hinter ELSE angegebenen Befehle durchgeführt.

INKEY$ (MSB)

```
Form: INKEY$
Beispiel: EINGABE$=INKEY$
```

Die Funktion INKEY$ liefert den ATASCII-Wert der zuletzt

gedrückten Taste. Dieser Befehl läßt sich in den anderen BASIC-Dialekten unschwer simulieren:

Durch OPEN#K,4,0,"K:" (wobei K die Nummer eines freien Kanals sein muß) wird zunächst ein Kanal für Eingabe von der Tastatur geöffnet. Mit GET#K,WERT:EINGABE$=CHR$(WERT) läßt sich dann ein Zeichen einlesen.

Anmerkungen

- Auf anderen Computersystemen wird die Funktion INKEY$ gewöhnlich benutzt, um festzustellen, welche Taste im Moment der Abfrage gedrückt ist. Die Programmausführung wird auf jeden Fall sofort fortgesetzt, auch wenn im Moment der Abfrage keine Taste gedrückt ist. Meist wird diese Funktion in Spielen verwendet, die durch die Tastatur gesteuert werden. Auf dem Atari empfiehlt es sich, entweder stattdessen den Joystick abzufragen (wordurch die Steuerung sowieso einfacher wird) oder durch „PEEK(764)" den internen Code der zuletzt gedrückten Taste zu ermitteln. Hier ist allerdings darauf zu achten, daß diese Werte nicht mit den ATASCII-Werten übereinstimmen.

INPUT

Abkürzung: I.
Form: INPUT (#nausdr:)varname(,varname...)
Beispiel: INPUT#E,ANTWORT

Mit Hilfe des INPUT-Befehls kann man numerischen und alphanumerischen Variablen Werte zuweisen. Dabei können die Daten entweder vom Bildschirm (dann braucht kein Kanal angegeben zu werden) oder von einem beliebigen anderen Periepheriegerät stammen.

Die Parameter

Als erster Parameter kann hinter INPUT die Nummer des Eingabekanals angegeben werden, den man dazu natürlich vorher geöffnet haben muß. Nur wenn kein Kanal angegeben ist, gibt das BASIC auf dem Bildschirm ein Fragezeichen aus. Man kann also auch Eingaben

vom Bildschirm ohne Ausgabe eines Fragezeichens machen, indem man einfach einen Kanal zum Lesen (→ OPEN) vom Bildschirm öffnet. Hinter der Kanalnummer muß ein Semikolon stehen. Alle weiteren Parameter sind die Namen der Variablen, denen die Werte zugewiesen werden sollen.

Arbeitsweise

Der INPUT-Befehl nimmt solange vom Eingabekanal ATASCII-Zeichen entgegen, bis ein RETURN-Zeichen (EOF – End Of File) vorkommt. Ist die angegebene Variable eine Zeichenkettenvariable, dann werden einfach alle angekommenen Zeichen in die Zeichenkette übertragen. Sind mehr Daten angekommen, als die Zeichenkettenvariable fassen kann, dann werden die überzähligen Zeichen einfach ignoriert. Hat man den Namen einer numerischen Variablen angegeben, dann werden die Zeichen als Ziffern einer Zahl interpretiert. Deshalb dürfen bei Eingaben von numerischen Variablen in der Eingabe nur Ziffern und die Zeichen +, – und E (Exponent) stehen.

Gleichzeitige Eingabe mehrerer Variablen

Gibt man hinter INPUT mehrere Variablennamen an, dann werden den Variablen nacheinander Werte zugewiesen. Handelt es sich nur um numerische Variablen, dann dürfen die Zahlenwerte durch Kommata getrennt eingegeben werden. Dies ist bei Zeichenkettenvariablen nicht möglich, da das Komma als Bestandteil der Zeichenkette interpretiert wird (dies gilt nicht für Microsoft-BASIC; in diesem BASIC-Dialekt muß man dazu den Befehl LINE INPUT verwenden). Um mehreren Zeichenketten Werte zuweisen zu können, muß man nach jeder Eingabe RETURN drücken (das Fragezeichen wird dann jedesmal neu ausgegeben).

Anmerkungen

- Manchmal ist es lästig, daß es im Atari-BASIC nicht ohne weiteres möglich ist, beim INPUT-Befehl das Fragezeichen zu unterdrükken. Es gibt jedoch einen einfachen Weg, um dieses Problem zu umgehen: Statt „INPUT“ verwendet man das Format „INPUT #kanal...“, nachdem man vorher den entsprechenden Kanal mittels „OPEN#kanal,12,0,“E:““ auf den Bildschirm geöffnet hat. Es wird dann exakt die gleiche Funktion ausgeführt wie beim

normalen INPUT-Befehl. Daneben hat die Benutzung dieser Befehlsform einen weiteren Vorteil: Programme die statt mit INPUT und PRINT mittels INPUT# und PRINT# arbeiten, können auch später noch sehr einfach auf die Benutzung anderer Peripheriegeräte umgeschrieben werden.

- Es ist leider nicht möglich, bei INPUT als Variablennamen den Namen einer Feldvariablen anzugeben. Stattdessen kann man, wie im folgenden Beispiel, eine andere Variable benutzen und deren Wert anschließend in die Feldvariable übertragen.

```
100 DIM LAENGE(100)
110 FOR I=1 TO 100
120 INPUT WERT
130 LAENGE(I)=WERT
140 NEXT I
```

INPUT...AT (MSB)

Form: INPUT #nausdr,AT(nausdr,nausdr)varname
Beispiel: INPUT#6,AT(7,9)wert

Mit INPUT...AT kann eine Variable an einer bestimmten „Koordinate" des Peripheriegeräts eingelesen werden. Das kann beispielsweise ein bestimmtes Byte eines Sektors der Diskettendatei oder auch ein Zeichen auf dem Graphikbildschirm sein.

INSTR (MSB)

Form: INSTR(nausdr,zvar,zvar)
Beispiel: STELLE=INSTR(1,DAT$,"Hallo")

Die Funktion sucht innerhalb der ersten Zeichenkette nach der zweiten. Mit dem ersten Argument kann man festlegen, bei dem wievielten Zeichen der Suchvorgang beginnen soll, um auch sämtliche mehrfach auftretende Zeichenketten finden zu können.

Wurde keine Übereinstimmung gefunden, so wird der Wert Null zurückgegeben.

In BASIC XL heißt die entsprechende Funktion FIND.

INT

Form: INT(nausdr)
Beispiel: A=INT(RND(0)*256)

Die INT-Funktion berechnet zu jeder Zahl die nächstkleinere ganze Zahl. Bei positiven Argumenten fallen dadurch nur die Nachkommastellen weg (Beispiel: INT (1,4151)=1. Im negativen Zahlenbereich ist jedoch die nächstkleinere ganze Zahl die Zahl, deren Betrag um eins höher ist, so daß „INT(−5.6)" −6 ist.

Anmerkungen

- Ein einfaches Verfahren, um Zahlen auf ganze Zahlen zu runden, ist, das Argument vor der Verwendung von INT um 0.5 zu erhöhen. Dadurch wird zum Beispiel 1.4 auf 1 und 1.6 auf 2 gerundet.

```
10 REM --------------------
20 REM Demonstration zu INT
30 REM --------------------
40 GRAPHICS 7+16:COLOR 1:SETCOLOR 2,0,
0
50 PLOT 80,0:DRAWTO 80,95
60 PLOT 0,48:DRAWTO 159,48
70 COLOR 2
80 FOR I=0 TO 159
90 PLOT I,48-INT((I-80)/10)*5
100 NEXT I
110 GOTO 110
```

Abb. I.1: INT – Beispielprogramm

Interface

Ein Interface ist ein Gerät, das den Computer mit einem anderen Peripheriegerät verbindet. So ist zum Beispiel zum Betrieb aller Drucker, die nicht von Atari selbst stammen, ein Druckerinterface wie das Atari 850 nötig.

Internationaler Zeichensatz

Der Atari ist, wie praktisch sämtliche anderen ausländischen Heimcomputer, im Grundzustand nicht dazu in der Lage, die deutschen Umlaute und das „ß“ darzustellen. Die XL-Geräte verfügen daher zusätzlich zum Standardzeichensatz, der neben den „normalen“ Zeichen auch Graphikzeichen umfaßt, auch über den sogenannten internationalen Zeichensatz, der außer den deutschen Umlauten auch Sonderzeichen aus anderen europäischen Sprachen enthält. Er läßt sich durch „POKE 756,204“ ein- und durch „POKE 756,224“ wieder ausschalten.

- Wie bei allen Neuerungen in den XL-Geräten sollte man bei der Anwendung möglichst auf irgendeine Art und Weise Rücksicht auf die Besitzer der älteren Modelle nehmen. Da dieser Zeichensatz das „ß“ sowieso nicht enthält, ist es für den fortgeschrittenen Programmierer vielleicht sinnvoller, einen selbst definierten Zeichensatz zu verwenden, wie es zum Beispiel auch in dem Textverarbeitungsprogramm „Atari-Schreiber“ geschehen ist.

J

Jiffies

Die interne Uhr des Ataris wird alle 1/50 Sekunden erhöht. Man nennt diesen Zeitabstand gewöhnlich Jiffie.

```
10 REM ------------------------
20 REM Demonstration zu Jiffies
30 REM ------------------------
40 UHR=18
50 REM Cursor ausblenden
60 POKE 752,1
70 REM Laufende Zeianzeige in Jiffies
80 PRINT :PRINT "          Seit Einscha
lten sind"
90 POSITION 10,8:PRINT "Jiffies vergan
gen."
100 POSITION 16,5:PRINT PEEK(UHR)*6553
6+PEEK(UHR+1)*256+PEEK(UHR+2)
110 GOTO 100
```

Abb. J.1: Jiffies – Beispielprogramm

```
  Seit Einschalten sind

        78849

Jiffies vergangen.
```

Abb. J.2: Jiffies – Beispiellauf

Joystick

Siehe STICK.



KILL (MSB)

Form: KILL zausdr
zausdr: Dateispezifikation
Beispiel: KILL "D:TEST.MSB"

Der Befehl KILL löscht Diskettendateien. Ist die Datei gesichert, wird es zum Fehler 167 kommen.

In BASIC XL heißt der entsprechende Befehl ERASE.

Kontrollzeichen

Neben den Ziffern, Buchstaben und Graphikzeichen kennt der Atari auch die sogenannten Kontrollzeichen, die normalerweise auf dem Bildschirm nicht sichtbar sind, sondern bestimmte Funktionen, wie das Bewegen des Cursors oder das Löschen eines Zeichens hervorrufen. Möchte man das entsprechende Zeichen auf den Bildschirm bringen, dann muß man vorher einmal die ESC-Taste und dann die entsprechende Tastenkombination drücken. So könnte man beispielsweise in eine Zeichenkette, die einen Programmtitel enthält, ein Bildschirmlöschzeichen integrieren, so daß der Bildschirm immer dann gelöscht wird, wenn diese Zeichenkette mit PRINT auf dem Bildschirm ausgegeben wird.

Daneben ist es auch möglich, nicht die Funktion, sondern das Zeichen an sich in eine Zeichenkette zu übernehmen. Dazu setzt man einfach vor das auszugebende Kontrollzeichen ein ESC-Zeichen, das man durch zweifaches Drücken der ESC-Taste erzeugen kann.

Wenn man viele Kontrollzeichen hintereinander ein- oder ausgeben möchte, bietet sich eine andere Methode an: Durch „POKE 766,1“

werden die Steuerfunktionen angeschaltet und die tatsächlichen Zeichen auf den Bildschirm gebracht. Zum normalen Funktionsmodus kann man durch „POKE 766,0“ oder Drücken der BREAK- oder SYSTEM-RESET-Taste zurückgelangen.

```
10 REM ------------------------------
-
20 REM Demonstration zu Kontrollzeiche
n
30 REM ------------------------------
-
40 REM Zeichensatz mit eingeschalteter

50 REM Steuerung durch Kontrollzeichen

60 FOR I=0 TO 255
70 PRINT CHR$(I);
80 NEXT I
90 REM ..und mit abgeschalteter Cursor
-
100 REM steuerung
110 POKE 766,1:PRINT
120 FOR I=0 TO 255
130 PRINT CHR$(I);
140 NEXT I
```

Abb. K.1: Kontrollzeichen – Beispielprogramm

```
•Kontrollzeichen bei ATARI-Computern•

  ␛ - Escape
  ↑ - Cursor nach oben
  ↓ - Cursor nach unten
  ← - Cursor nach links
  → - Cursor nach rechts
  ↖ - Bildschirm loeschen
  ◀ - Zeichen vor dem Cursor loeschen
  ▶ - Tabulator
  ⊡ - Zeile loeschen
  ⊟ - Zeile einfuegen
  ⊠ - Tabulator loeschen
  ⊞ - Tabulator setzen
  ⍰ - Warnton ('Buzzer')
  ◧ - Zeichen nach dem Cursor loeschen
  ◨ - Zeichen einfuegen
```

Abb. K.2: Tabelle der Kontrollzeichen

K-Byte

Oft wird im Zusammenhang mit Speichergröße der Begriff „K-Byte“ oder „Kilo-Byte“ verwendet. Bei einem K-Byte handelt es sich nicht, wie oft fälschlicherweise angenommen wird, um 1000 Bytes, sondern um 1024 Bytes.

L

LEFT$ (MSB, BASIC XL)

Form: LEFT$(zausdr,nausdr)
zausdr: Zeichenkette
nausdr: Anzahl der Zeichen
Beispiel: PRINT LEFT$(DAT$,6)

Die Funktion LEFT$ liefert eine Teilzeichenkette, die am Anfang der durch den ersten Parameter bestimmten Zeichenkette beginnt und die eine durch den zweiten Parameter angegebene Länge hat. In diesem Fall würden also die ersten sechs Zeichen der Zeichenkette DAT$ ausgegeben.

- Im Atari-BASIC könnte man durch PRINT DAT$(1,6) das gleiche Ergebnis erreichen.

LEN

Form: LEN (zausdr)
Beispiel: LAENGE=LEN(DAT$)

Die Funktion LEN bestimmt die Länge einer Zeichenkette. Das Argument darf dabei sowohl eine Zeichenkette als auch eine Zeichenkettenvariable sein.

- Es ist zu beachten, daß die Länge einer Zeichenkette nicht mit der bei DIM angegebenen Höchstlänge übereinstimmen muß. Eine definierte Länge hat sie erst dann, wenn ihr durch irgendeine Anweisung ein Inhalt zugewiesen worden ist.

```
10 REM --------------------
20 REM Demonstration zu LEN
30 REM --------------------
40 REM Zeichenkette initialisieren
50 DIM NAME$(40)
60 REM Namen eingeben
70 PRINT "Wie heisst Du";
80 INPUT NAME$
90 REM Laenge berechnen und Namen
100 REM mittig ausgeben
110 L=LEN(NAME$)
120 PRINT "K":REM Bildschirm loeschen
130 POSITION 20-L/2,4:PRINT NAME$
140 END
```

Abb. L.1: LEN – Beispielprogramm

```
Wie heisst Du?Julian Reschke




                Julian Reschke
```

Abb. L.2: LEN – Beispiellauf

LET

Abkürzung: LE. oder durch Weglassen
Form: LET nvar=nausdr
LET nvar=logausdr
LET zvar=zausdr
Beispiel: LET Q=2*PI

Der Befehl LET dient dazu, der hinter LET stehenden Variablen den neuen Wert, der hinter den Gleichheitszeichen steht, zuzuordnen. Hinter dem Gleichheitszeichen darf dabei ein beliebiger Aus-

druck stehen. Es muß jedoch beachtet werden, daß Zeichenketten-variablen nur alphanumerische Werte und numerischen Variablen nur numerische Werte zugeordnet werden können.

- Die beiden Datentypen lassen sich durch VAL, ASC, CHR$ und STR$ untereinander in Beziehung bringen.
- Wie auf praktisch allen gängigen Computersystemen (Ausnahme: Sinclair) darf man in den verschiedenen BASIC-Dialekten des Ataris das Kommando LET weglassen. Schwierigkeiten ergeben sich nur dann, wenn die Anfangsbuchstaben einer Variablen mit dem Namen eines Kommandos oder einer Funktion übereinstimmen (Beispiel: ADRESSE=ADR(„Text“)). In diesem Fall darf man also das Wort LET nicht weglassen.

LINE INPUT (MSB)

Form: LINE INPUT (#nausdr)("Text";)varname(,varname...)
Beispiel: LINE INPUT "Eingabe ";WERT

Bei der Benutzung von LINE INPUT ist es möglich, auch Anführungsstriche, Kommata oder ein Semikolon miteinzugeben.

LINE INPUT...AT (MSB)

(Siehe INPUT...AT und LINE INPUT).

LIST

Abkürzung: L.
Form: LIST ((zausdr,)nausdr1(,nausdr2))
zausdr: Gerätename
nausdr1: Anfangszeile
nausdr2: Endzeile
Beispiel: LIST "D:PROG.LST",100,199

Der Befehl LIST dient dazu, den Programmtext in lesbarer Form auszugeben. Dabei wird das im Speicher befindliche BASIC-Pro-

gramm in lesbare Buchstaben verwandelt. Die ausgegebenen Zeilen gleichen bis auf zwei Ausnahmen dem eingetippten Programmtext:

- Zwischen alle Befehle und Funktionen werden automatisch Leerzeichen eingesetzt.
- Abkürzungen werden durch die vollen Befehlsnamen ersetzt.

- In BASIC XL wird das Programm in etwas übersichtlicherer Form ausgegeben: Bei FOR-NEXT-Schleifen, WHILE-ENDWHILE-Schleifen und IF-Konstruktionen werden die „innen" stehenden Zeilen eingerückt, um die Programmstruktur zu verdeutlichen. Außerdem werden alle auf das erste Zeichen folgenden Buchstaben in Kleinschrift ausgegeben. Beide Unterschiede lassen sich mit Hilfe des Befehls SET beseitigen.

Gibt man kein Argument an, dann wird einfach das gesamte Programm auf dem Bildschirm ausgegeben. Gibt man als Argument einen einzigen alphanumerischen Wert an, wird nur die betreffende Zeile, sofern vorhanden, ausgegeben. Zusätzlich dazu kann, durch ein Komma abgetrennt, auch das Ende des auszugebenden Bereiches festgelegt werden.

Daneben kann als erstes Argument auch noch eine Dateispezifikation angegeben werden. In diesem Fall wird das Listing auf dem angegebenen Gerät statt auf dem Bildschirm ausgegeben. Im obengenannten Beispiel würden also die Zeilen von 100 bis 199 einschließlich auf der Diskette unter dem Namen „PROG.LST" abgespeichert (LST ist die gebräuchliche Namenserweiterung für durch LIST abgespeicherte Programme).

Über das gleiche System kann man mit LIST"P:" eine Ausgabe des BASIC-Programms auf dem Drucker bewirken.

LOAD

Abkürzung: LO.
Form: LOAD zausdr
zausdr: Gerätename
Beispiel: LOAD "D:BEISPIEL.BAS"
LOAD DATEI$

Mit dem Befehl LOAD kann man ein vorher mit SAVE abgespeichertes BASIC-Programm wieder in den Speicher laden. Als Parameter muß man die Dateispezifikation selbst oder eine Zeichenkettenvariable, die sie enthält, angeben. Im Prinzip kann man alle Gerätenamen angeben, obwohl eigentlich nur die Diskettenstation ("D:NAME.ERW") und der Kassettenrecorder ("C:") sinnvolle Resultate ergeben. Beim Laden von Kassette muß folgendermaßen vorgegangen werden: Nach Eingabe von LOAD "C:" ertönt ein Brummton, woraufhin auf dem Kassettenrecorder PLAY und auf der Tastatur eine alphanumerische Taste gedrückt werden muß. Daraufhin beginnt der Ladevorgang, der wie auch alle anderen Funktionen des Kassettenrecorders von Kontrolltönen begleitet wird.

- Möchte man, daß ein Programm von Kassette automatisch das folgende Programm nachlädt, ohne daß eine Taste gedrückt werden muß, kann man vor den LOAD-Befehl „POKE 764,12" setzen. Dadurch wird das Tastaturregister dahingehend verändert, daß der Computer annimmt, es sei bereits eine Taste gedrückt worden.
- Wird der Ladevorgang durch einen Fehler, BREAK oder SYSTEM-RESET unterbrochen, dann kann der bisher geladene Programmteil nicht gerettet werden.
- Weitere Hinweise über die Struktur von LOAD-Dateien finden sich unter SAVE.

LOCATE (Atari, BASIC XL)

Abkürzung: LOC.
Form: LOCATE nausdr1,nausdr2,nvar
nausdr1: X-Position
nausdr2: Y-Position
nvar: Name der Variablen, in die das Ergebnis übertragen werden soll
Beispiel: LOC. SPALTE,REIHE,INHALT

LOCATE ist ein Befehl zur Bestimmung der Farbe (in Graphikstufen) eines Punktes oder des ATASCII-Wertes (in Textstufen) eines

Zeichens an einer bestimmten Bildschirmposition. Der Arbeitsweise nach müßte LOCATE eigentlich eine Funktion sein, da es einen Wert bestimmt und diesen in eine numerische Variable überträgt. Im einzelnen müssen als erste beide Parameter die Koordinaten der abzufragenden Bildschirmposition angegeben werden. Das dritte

```
10 REM ------------------------
20 REM Demonstration zu LOCATE
30 REM ------------------------
40 GRAPHICS 24:SETCOLOR 2,0,0:COLOR 1
50 FOR I=0 TO 1.5708 STEP 0.1
60 PLOT 160,96:DRAWTO 160+150*SIN(I),9
6+90*COS(I)
70 PLOT 160,95:DRAWTO 160+150*SIN(I),9
5-90*COS(I)
80 PLOT 159,95:DRAWTO 159-150*SIN(I),9
5-90*COS(I)
90 PLOT 159,96:DRAWTO 159-150*SIN(I),9
6+90*COS(I)
100 NEXT I
110 FOR I=0 TO 319
120 K=0
121 FOR J=0 TO 191
130 LOCATE I,J,WERT:IF WERT THEN K=(K=
0)
140 IF K THEN PLOT I,J
150 NEXT J
160 NEXT I
170 REM auf BREAK warten
180 GOTO 170
```

Abb. L.3: LOCATE – Beispielprogramm

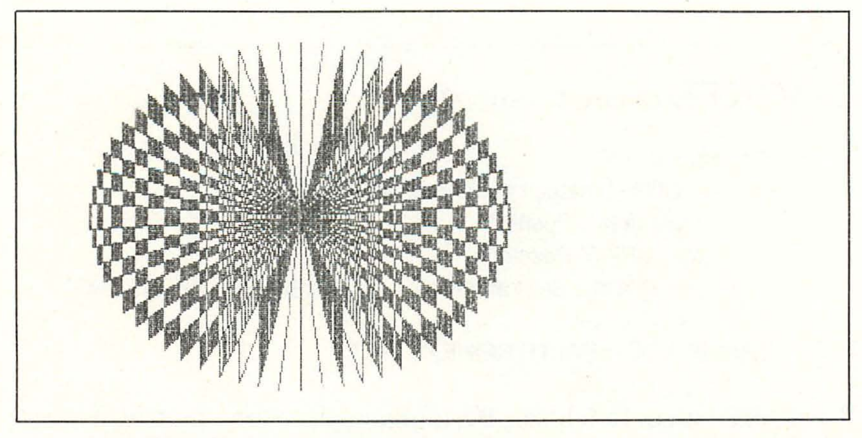


Abb. L.4: LOCATE – Beispiellauf

Argument ist der Name der Variablen, in die der ermittelte Wert übertragen werden soll. Im obigen Beispiel würde also die Variable INHALT den Wert der Farbe des Punktes oder den ATASCII-Wert des Zeichens an der Bildschirmposition REIHE, SPALTE annehmen.

LOCK (MSB)

Form: LOCK zausdr
zausdr: Dateispezifikation
Beispiel: LOCK DATEI

Mit LOCK können auf der Diskette abgespeicherte Dateien gegen Veränderungen wie Umbenennung oder Löschung gesichert werden.

- Der Befehl PROTECT in BASIC XL hat die gleiche Funktion

LOG

Form: LOG (nausdr)
Beispiel: W=LOG(A)

```
10 REM --------------------
20 REM Demonstration zu LOG
30 REM --------------------
40 REM hochaufloesende Graphik
50 GRAPHICS 24
60 REM Hintergrund schwarz, in weiss
70 REM zeichnen
80 SETCOLOR 2,0,0:COLOR 1
90 PLOT 0,0:DRAWTO 0,191:DRAWTO 319,19
1:PLOT 0,191
100 REM Logarithmuskurve zeichnen
110 FOR I=1 TO 319 STEP 3
120 DRAWTO I,191-31*LOG(I)
130 NEXT I
140 REM ...und auf BREAK warten
150 GOTO 140
```

Abb. L.5: LOG – Beispielprogramm

Die Funktion LOG berechnet den Logarithmus einer Zahl zur natürlichen Basise.

Ein Verfahren zur Berechnung des Logarithmus zu einer beliebigen Basis wird unter CLOG beschrieben.

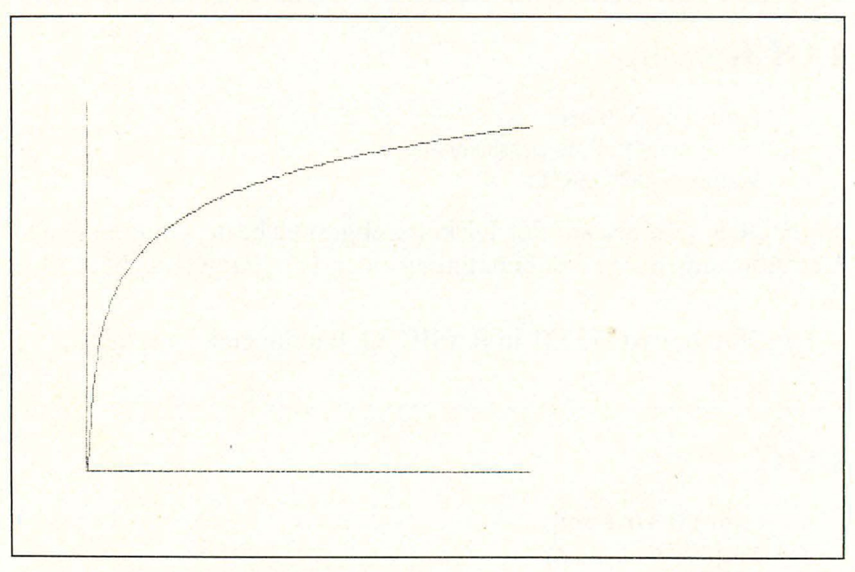


Abb. L.6: LOG – Beispiellauf

Logischer Ausdruck

Ein logischer Ausdruck kann eigentlich nur zwei Werte annehmen: „wahr" und „falsch" (true, false). Da BASIC im Gegensatz zu Sprachen wie Pascal diesen Datentyp nicht kennt, wird dieser durch numerische Werte dargestellt. Dabei entspricht der Wert „falsch" der Zahl Null und der Wert „wahr" allen Zahlen außer Null.

Aus dieser Beziehung zwischen logischen und numerischen Werten ergibt sich die interessante Folge, daß man in gewissem Umfang logische und numerische Ausdrücke mischen kann.

Sehen wir uns doch mal einen gewöhnlichen IF-Ausdruck an:

```
IF FLAG<>0 THEN 1000
```

In dieser Zeile steckt der logische Ausdruck „FLAG<>0". Dieser Ausdruck hat den Wert „falsch", wenn FLAG den Inhalt „Null" hat. Da der logische Wert „falsch" ebenfalls den numerischen Wert „Null" hat, liegt es nahe, statt dessen einfach

```
IF FLAG THEN 1000
```

zu schreiben, was nicht nur kürzer, sondern auch noch übersichtlicher und verständlicher ist. Mit Hilfe dieser Methode läßt sich also die Form „nausdr<>0" ersetzen.

Ebenso kann man mit logischen Ausdrücken der Form „nausdr=0" verfahren, wenn man sie durch „NOT nausdr" ersetzt.

Doch auch in der entgegengesetzten Richtung kann eine Mischung stattfinden. So läßt sich zum Beispiel die Zeile

```
IF B=1 THEN A=A+1
```

auch folgendermaßen schreiben:

```
A=A+(B=1).
```

LOMEN (BASIC XL)

Abkürzung: LOM.
Form: LOMEM nausdr
nausdr: neue Anfangsadresse des Programmspeichers
Beispiel: LOMEM: ALTERWERT+1024

Mit LOMEM kann eine neue Anfangsadresse des BASIC-Programmspeichers festgelegt und so freier Speicherplatz „unter" dem BASIC-Programm geschaffen werden.

Vorsicht: Bei Ausführung des Kommandos LOMEM wird das im Speicher befindliche BASIC-Programm vernichtet!

LPRINT (Atari BASIC, BASIC XL)

Abkürzung: LP.
Form: LPRINT

LPRINT erfüllt die gleiche Aufgabe wie PRINT, nur daß die Ausgabe statt dessen auf dem Drucker erfolgt.

Als Ausgabekanal wählt BASIC automatisch Kanal 7 aus, so daß Fehler auftreten können, wenn man diesen Kanal auch noch für andere Zwecke benutzt.

Ein besserer Weg, Texte auf dem Drucker auszugeben, ist die Benutzung des Befehls PRINT#kanal.

LVAR (BASIC XL)

Abkürzung: LV.
Form: LVAR (zausdr)
zausdr: Dateispezifikation
Beispiel: LVAR
LVAR "P:"

Dieser Befehl ermöglicht es, eine Liste zu erzeugen, die die Namen aller Variablen und die Zeilennummern, in denen sie auftreten, enthält. Zeichenkettenvariablen sind dabei durch ein angehängtes Dollarzeichen ($) und Feldvariablen durch eine geöffnete Klammer gekennzeichnet.

Als Argument kann der Name des Peripheriegeräts, auf dem die Liste ausgegeben werden soll, angegeben werden. Setzt man kein Argument dazu, dann wird die Liste automatisch auf den Bildschirm ausgegeben.



Menü

Menü ist die Bezeichnung für eine spezielle Art des Aufbaus von Programmen. Bei einem menügesteuertem Programm wird auf dem Bildschirm eine Liste der abrufbaren Programmteile ausgegeben, aus der dann einer durch Eingabe einer Kennziffer oder eines Kennbuchstabens o. ä. ausgewählt wird. Beispiele für menügesteuerte Programme sind das DOS oder der Atari-Schreiber. Menügesteuerte Programme bieten den Vorteil, daß man das betreffende Programm auch dann bedienen kann, wenn man es nicht genau kennt. Damit ist diese Art von Programmierung gerade für den unerfahrenen Benutzer sehr angenehm.

Kennt man ein Programm mit Menüsteuerung jedoch erst einmal genauer, dann wird man sich wohl über die umständliche Bedienung und den hohen Speicherplatzbedarf der Menüs ärgern.

Das Gegenteil von Menüsteuerung ist die Steuerung durch Kommandoeingabe, die beispielsweise in jedem BASIC angewendet wird. Das Programm (hier also: die Sprache BASIC) wird dabei durch die Eingabe bestimmter Kommandos wie RUN oder LIST gesteuert.

Ein optimaler Kompromiß ist es, dem Benutzer die Auswahl zwischen Menü- und Kommandosteuerung zu ermöglichen. Der Anfänger kann sich dann mit Menüs durch das für ihn unbekannte Programm leiten lassen, während der Fortgeschrittenere die Kommandoeingabe wählen wird, um schneller arbeiten zu können. Ein gutes Beispiel für solch ein Programm ist DOS XL von OSS.

MERGE (MSB)

Form: MERGE zausdr
zausdr: Dateiname des anzuhängenden Programms
Beispiel: MERGE "D:HILFE.LST"

Entspricht dem Atari-BASIC-Befehl ENTER.

Maschinensprache

„Maschinensprache" ist die einzige Computersprache, die von einem Mikroprozessor ohne weiteres verstanden werden kann. Deshalb können auch in Maschinensprache geschriebene Programme schneller ausgeführt werden als Programme in allen anderen Sprachen. Daneben bietet Maschinensprache noch andere Vorteile:

- Weil keine andere Programmiersprache zusätzlich benötigt wird, ist der übrigbleibende Speicherplatz größer;

- Maschinensprache ist die einzige Computersprache, in der der Programmierer eine vollständige Kontrolle (und Verantwortung) über die Speicherverwaltung hat.

Maschinensprache besteht eigentlich nur aus einzelnen Bytes, die je nach Kontext als Befehle, Parameter oder Daten interpretiert werden. Wenn man direkt in Maschinensprache programmieren wollte, müßte man jeden einzelnen Befehl anhand einer Tabelle in eine Codezahl übersetzen. Da man diese Übersetzung vor jedem Probelauf eines Programms vornehmen müßte, würde diese Art der Programmierung sehr viel Zeit in Anspruch nehmen. Einfacher ist es, einen sogenannten Assembler wie AMAC von Atari oder MAC/65 von OSS zu verwenden, der es erlaubt, Maschinensprachbefehle als normale Befehlsworte (sogenannte Mnemonics) einzutippen, die daraufhin automatisch von Assembler in ausführbaren Maschinencode übersetzt werden.

MID$ (BASIC XL, MSB)

Form: MID$(zausdr,nausdr1,nausdr2)
zausdr: zu bearbeitende Zeichenkette
nausdr1: erstes Zeichen
nausdr2: Anzahl der Zeichen
Beispiel: MID$("BASIC-Handbuch",7,4)

Mit der Funktion MID$ kann aus einer Zeichenkette eine Teilzeichenkette extrahiert werden. Dazu werden außer der Zeichenkette die Nummer des ersten Zeichens und die Anzahl der Zeichen angegeben.

Im obenstehenden Beispiel würde damit die Zeichenkette „Hand" bestimmt werden.

- Im Atari-BASIC könnte man zur Nachahmung dieser Funktion folgendes schreiben:

```
Z$="BASIC-Handbuch"
?Z$(7,11)
```

MISSILE (BASIC XL)

Abkürzung: MIS.
Form: MISSILE nausdr1,nausdr2,nausdr3
nausdr1: Nummer des Missiles
nausdr2: Vertikale Position
nausdr3: Höhe des Missiles
Beispiel: MIS. 0,50,2

Der Befehl MISSILE erlaubt es, das Missile eines bestimmten Players einzuschalten. Dazu muß die Nummer des Missiles, die vertikale Position sowie die Höhe angegeben werden. Das Missile kann dann mit dem Befehl PMMOVE über den Bildschirm bewegt werden.

Da beim Setzen des Missiles nicht einfach die entsprechenden Bits im Player-Missile-Speicher gesetzt werden, sondern umgeschaltet werden, kann man das Missile durch nochmaliges Setzen an der gleichen Bildschirmposition wieder löschen.

MOVE (BASIC XL)

Abkürzung: M.
Form: MOVE nausdr1,nausdr2,nausdr3
nausdr1: Startadresse
nausdr2: Zieladresse
nausdr3: Anzahl der Bytes
Beispiel: MOVE 57344,ADR(FONT$),1024

Dieser Befehl dient zum Verschieben großer Byteblöcke. Dabei muß man aber vorsichtig sein, da nicht überprüft wird, ob durch den Verschiebevorgang wichtige Daten zerstört werden.



NAME...TO (MSB)

Form: NAME zausdr TO zausdr
Beispiel: NAME"D:*.CMD" TO "*.COM"

Mit diesem Befehl können Dateinamen auf der Diskette verändert werden. Im oben angeführten Beispiel würde z. B. bei allen Dateien die Namenserweiterung „CMD" durch „COM" ersetzt. Im Zeichenkettenausdruck für den neuen Programmnamen braucht übrigens der Gerätename nicht noch einmal angegeben zu werden.

NEW

Form: NEW
Beispiel: NEW

Der Befehl NEW löscht das im Speicher befindliche BASIC-Programm.

- Nach Ausführung von NEW kehrt BASIC sofort in den Eingabemodus zurück, so daß nach NEW stehende Befehle nicht mehr ausgeführt werden.

NEXT

Abkürzung: N.
Form: NEXT nvar
Beispiel: N.I

NEXT beendet eine FOR-NEXT-Schleife. Als Argument muß der Name der Schleifenvariablen angegeben werden.

Weiteres dazu unter FOR.

NOT

Form: NOT logausdr
Beispiel: IF NOT X>Y THEN GOTO 10

Mit Hilfe des logischen Operators NOT kann der Wahrheitswert des angegebenen logischen Ausdrucks umgekehrt werden, das heißt: aus unwahr wird wahr und umgekehrt.

logausdr	NOT logausdr
wahr	unwahr
unwahr	wahr

Da alle denkbaren Vergleichsoperationen in Atari-BASIC verfügbar sind, ist der Operator NOT eigentlich überflüssig. Allerdings lassen sich Programme oft durch Anwendung von NOT übersichtlicher, kürzer und lesbarer gestalten.

NOTE

Abkürzung: NO.
Form: NOTE #nausdr,nvar1,nvar2
nausdr: Kanalnummer
nvar1: Variable, in die die Sektornummer übertragen wird
nvar2: Variable, in die die Position innerhalb des Sektors übertragen wird
Beispiel: NOTE#2,SEKTOR,BYTE

NOTE dient zur Bestimmung der aktuellen Schreibposition der durch die Kanalnummer spezifizierten Diskettendatei. Dabei wird die Sektornummer der ersten Variablen und die Bytenummer der

zweiten Variablen zugewiesen. Dieser Befehl kann benutzt werden, um die Position eines bestimmten Datensatzes auf der Diskette festzustellen, um dann später mit POINT wieder auf ihn zugreifen zu können.

NUM (BASIC XL)

Abkürzung: NU.
Form: NUM nausdr1,nausdr2
nausdr1: Anfangszeile
nausdr2: Schrittweite
Beispiel: NUM 1000,10

Der Befehl NUM kann bei der Erstellung längerer Programme sehr viel Zeit einsparen. Als erstes Argument ist anzugeben, bei welcher Programmzeile der einzugebende Bereich beginnt; das zweite Argument setzt dazu die Schrittweite fest. Nach Eingabe dieses Befehls erzeugt BASIC XL die Zeilennummern automatisch, so daß nur noch die Befehle eingegeben werden müssen.

Erreicht man dabei die Nummer einer bereits existierenden Programmzeile, dann wird die automatische Zeilennummernerzeugung abgebrochen.

- Auch das Microsoft-BASIC bietet eine solche Eingabehilfe an: den Befehl AUTO.



ON ERROR GOTO (MSB)

Form: ON ERROR GOTO nausdr
nausdr: Zeilennummer
Beispiel: ON ERROR GOTO 1000

Dieses Kommando entspricht TRAP in Atari-BASIC und BASIC XL.

ON...GOSUB

Abkürzung: nicht möglich
Form: ON nausdr GOSUB nausdr(,nausdr...)
Beispiel: ON WAHL GOSUB 100,300,350

Mit Hilfe des ON-Kommandos kann man Unterprogramme von dem Wert eines numerischen Ausdrucks abhängig aufrufen. Dabei wird das Unterprogramm an der ersten angegebenen Zeilennummer aufgerufen, wenn der Wert des numerischen Ausdrucks „1“ ist usw. Wird keine dem Zahlenwert entsprechende Zeilennummer gefunden, dann wird das Programm beim folgenden Befehl fortgesetzt.

ON...GOTO

Entspricht ON...GOSUB mit dem Unterschied, daß die Zeilennummern nicht durch einen GOSUB-Befehl, sondern durch einen GOTO-Befehl aufgerufen werden.

OPEN (Atari BASIC, BASIC XL)

Abkürzung: O.
Form: OPEN #nausdr1,nausdr2,nausdr3,zausdr
nausdr1: Kanalnummer
nausdr2: Kommando
nausdr3: zusätzliche Parameter zum Kommando
zausdr: Gerätespezifikation
Beispiel: OPEN#2,6,0,"D:*.*"

Mit dem Befehl OPEN kann ein Ein- und Ausgabekanal geöffnet werden.

Erstes Argument: die Kanalnummer

Als Kanalnummer können Zahlen zwischen 1 und 7 angegeben werden. Dabei muß aber beachtet werden, daß Kanal 6 für Graphik und Kanal 7 für verschiedene Disketten-, Kassetten- und Druckerfunktionen verwendet wird. Kanal 0 wird für den Textbildschirm verwendet und kann unter Atari-BASIC (im Gegensatz zu Basic XL) nicht benutzt werden.

Zweites Argument: Kommando

Mit Hilfe der zweiten Zahl wird festgelegt, welche Funktion auf dem Kanal ausgeführt werden soll.

Drittes Argument: Zusätzliche Angaben zur Funktion

Die dritte Zahl muß meistens auf 0 gesetzt werden; Abweichungen davon werden in den folgenden Absätzen angeführt.

Viertes Argument: Gerätename

Hier können die Namen aller verfügbaren Peripheriegeräte angegeben werden. Bei Verwendung der Diskettenstation muß dann noch der Dateiname dazugesetzt werden. Es sollte übrigens beachtet werden, daß nicht jedes Gerät auch jede Funktion ausführen kann.

```
10 REM ---------------------
20 REM Demonstration zu OPEN
30 REM ---------------------
40 REM Zeichenketten initialisieren
50 DIM GERAETE$(14)
60 DIM TASTEN$(1)
70 DIM DATEN$(FRE(0)-1000)
80 REM Befehlseingabe
90 PRINT "Lesen oder Schreiben";
100 INPUT TASTEN$
110 IF TASTEN$="L" OR TASTEN$="S" THEN
 140:REM Eingabe korrekt
120 PRINT "□":REM Piepton
130 GOTO 90:REM Eingabe wiederholen
140 REM Geraetenamen eingeben
150 PRINT
160 PRINT "Geraetename";
170 INPUT GERAETE$
180 REM Funktion ausfuehren
190 IF TASTEN$="L" THEN GOSUB 1000
200 IF TASTEN$="S" THEN GOSUB 1100
210 GOTO 90
999 REM Lesen
1000 OPEN #1,4,0,GERAETE$
1010 TRAP 1060:REM Dateiende abfangen
1020 LAENGE=0
1030 GET #1,WERT:LAENGE=LAENGE+1
1040 DATEN$(LAENGE)=CHR$(WERT)
1050 GOTO 1030:REM naechstes Byte lese
n
1060 CLOSE #1:TRAP 40000
1070 RETURN
1099 REM Schreiben
1100 OPEN #1,8,0,GERAETE$
1110 FOR I=1 TO LAENGE
1120 PUT #1,ASC(DATEN$(I))
1130 NEXT I:CLOSE #1
1140 RETURN
```

Abb. O.1: OPEN – Beispielprogramm

```
Lesen oder Schreiben?L

Geraetename?D:*.LST
Lesen oder Schreiben?C:

Lesen oder Schreiben?S

Geraetename?Q:

ERROR-    130 AT LINE 1100
```

Abb. O.2: OPEN – Beispiellauf

OPEN und der Kassettenrecorder ("C:")

Im Zusammenhang mit dem Kassettenrecorder können Daten entweder nur gelesen oder nur geschrieben werden. Dem dritten Parameter kommt beim Schreiben auf Kassette eine besondere Bedeutung zu: Ist er „0", dann werden die Datenblöcke durch normal lange Pausen untereinander getrennt. Gibt man aber statt dessen „128" an, sind diese Aufzeichnungspausen wesentlich kürzer.

OPEN und die Diskettenstation ("Dn:NAME.ERW")

Um den Befehl OPEN im Zusammenhang mit der Diskettenstation benutzen zu können, muß man vorher das Diskettenbetriebssystem (DOS) gebootet haben. Da es für die Diskettenstation eine besonders große Anzahl von Betriebsmodi gibt, werden sie im folgenden einzeln erklärt.

Datei lesen (Kommando: 4)

Möchte man eine Datei einlesen, dann muß als zweiter Parameter die Zahl „4" benutzt werden. Der Name der Diskettendatei braucht nicht vollständig angegeben zu werden (siehe „Wild-Cards").

Directory lesen (Kommando: 6)

Mit Hilfe des Befehls 6 können die Dateinamenseintragungen als normale Datei eingelesen werden. Dabei muß als Gerätename der „Suchbegriff" angegeben werden. (Um alle Dateinamen einzulesen, müßte man demzufolge "D:*.*" verwenden.) Der letzte Eintrag ist stets die Meldung „... FREE SECTORS". Daher werden auch dann Daten eingelesen, wenn die „gesuchte" Datei nicht auf der Diskette ist.

- Ein einfaches Programm zur Ausgabe des Directorys finden Sie unter „Directory".

Ermittelt man mit Hilfe dieses Befehls die Dateinamen aus dem Directory, dann liegen sie nicht in der üblichen Form vor, sondern müssen erst umgewandelt werden. Dazu kann man das folgende kurze Programm benutzen:

```
10 REM -------------------------
20 REM Umwandlung von Dateinamen
30 REM (siehe OPEN#n,6,0,datein.
40 REM -------------------------
50 REM 2 Zeichenketten bereitstellen
60 DIM FN1$(20),FN2$(20)
70 REM Dateinamen aus Directory
80 REM lesen
90 GOSUB 1000
100 PRINT "Vorher: ";FN1$
110 REM ...und umwandeln
120 GOSUB 1100
130 PRINT "Nachher: ";FN1$
140 END
999 REM Dateinamen einlesen
1000 CLOSE #1:OPEN #1,6,0,"D:*.*"
1010 INPUT #1;FN1$
1020 CLOSE #1
1030 RETURN
1099 REM Dateinamen umwandeln
1100 FN2$="D:"
1105 FN2$(3)=FN1$(3)
1110 FOR I=3 TO 10
1120 IF FN2$(I,I)=" " THEN POP :GOTO 1
140
1130 NEXT I
1140 FN2$(I)="."
1150 FN2$(I+1)=FN1$(11,13)
1160 FN1$=FN2$
1170 RETURN
```

Abb. O.3: OPEN #n – Beispielprogramm

```
Vorher:   DOS      SYS 043
Nachher:  D:DOS.SYS
```

Abb. O.4: OPEN #n – Beispiellauf

Datei schreiben (Kommando: 8)

Auch beim Schreiben von Diskettendateien braucht der Dateiname nicht vollständig angegeben zu werden. Gibt man beispielsweise "D:*.BAS" an, dann wird das erste Programm mit der Namenser-

weiterung „.BAS“ gesucht, gelöscht und neu beschrieben. Man sollte übrigens bei der Anwendung von „Wild-Cards“ beim Schreiben von Dateien sehr vorsichtig sein.

Datei erweitern (Kommando: 9)

Diese Funktion erlaubt es, Daten an eine bestehende Datei „anzuhängen“. Dabei werden die neu geschriebenen Daten einfach hinter das Ende der ursprünglichen Datei gehängt. Beim Anfügen von Daten wird stets die gesamte Datei nochmals gelesen, weil zuerst das Ende der Datei gefunden werden muß.

Datei aktualisieren (Kommando: 12)

Bei Verwendung der Zahl „12“ als zweiter Parameter, kann eine Datei gleichzeitig eingelesen und neu beschrieben werden.

- Weil es nicht möglich ist, auf normale Art und Weise das DOS auf eine Diskette zu schreiben, existiert dafür folgende Sonderfunktion: “OPEN #kanal,8,0,„D:DOS.SYS“:CLOSE#kanal“. Diese beiden Befehle reichen aus, das Diskettenbetriebssystem auf eine Diskette zu schreiben.

Das Diskettenbetriebssystem kann im Grundzustand gleichzeitig maximal drei offene Dateien verwalten. Unter DOS 2.0S ist es möglich, diese Anzahl mit dem folgenden Programm bis auf sieben zu erhöhen. Die Vergrößerung dieser Zahl bedeutet jedoch auch gleichzeitig eine Verringerung des zur Verfügung stehenden Speicherplatzes.

OPEN und der Drucker (“P:“)

Der Drucker kann nur zur Ausgabe geöffnet werden (Parameter 8). Auf dem nicht mehr vertriebenen ATARI-Drucker 820 konnte man durch Angabe von „83“ als dritter Parameter die Ausgabe auf liegende Zeichen umschalten.

OPEN und die RS-232-Schnittstelle (“Rn:“)

Für die serielle Schnittstelle stehen die Kommandos Lesen (5), Schreiben in Blöcken (8), normales Schreiben (9) und gleichzeitiges Lesen und Schreiben von Blöcken (13) zur Verfügung.

OPEN und der Graphikbildschirm ("S:")

Ein OPEN-Befehl auf dem Graphikbildschirm ("S:") entspricht einem GRAPHICS-Befehl. ...?

OPEN und die Tastatur ("K:")

Die Tastatur kann nur als Eingabegerät (Parameter 4) verwendet werden.

OPEN und der Textbildschirm ("E:")

Der Editor ist eigentlich kein Ein- und Ausgabegerät im engeren Sinne, da er als Eingabemedium die Tastatur und als Ausgabemedium den Bildschirm ("S:") benutzt. Damit sind die Standardkommandos 4, 8 und 12 verfügbar.

Außerdem kann man auch (Kommando: 13) den Bildschirm zur Eingabequelle erklären. Damit wird es möglich, Eingaben automatisch erfolgen zu lassen.

OPTION (MSB)

Form: OPTION Kommando
Kommando: BASE, CHR, PLM oder RESERVE
Beispiel: OPTION RESERVE 20

Dies ist ein Vielzweckkommando, mit dem freier Speicherplatz für verschiedene Anwendungen wie Maschinensprachprogramme, Zeichensätze oder Player-Missile-Graphik, reserviert werden kann.

OR

Form: logausdr1 OR logausdr2
Beispiel: IF A=12 OR B=C+D THEN 3000

Der logische Operator OR dient zur Verknüpfung zweier einzelner logischer Ausdrücke. Das Ergebnis des Gesamtausdrucks wird nur dann „unwahr", wenn beide Einzelbedingungen „unwahr" sind.

Sobald auch nur eine der beiden Zusagen zutrifft, wird der Gesamtwert der Aussage „wahr“.

logausdr1	logausdr2	logausdr1 OR logausdr2
wahr	wahr	wahr
unwahr	wahr	wahr
wahr	unwahr	wahr
unwahr	unwahr	unwahr



PADDLE

Form: PADDLE (nausdr)
nausdr: Nummer des Dehreglers
Beispiel: P=PADDLE(0)

Die Funktion PADDLE dient zum Lesen der Stellung der Drehregler. Dabei wird als Parameter (0 bis 7) die Nummer des angeschlossenen Drehreglers angegeben. Der Funktionswert liegt stets zwischen 1 und 228.

- Da die neuen XL-Modelle nur noch über zwei Joystickanschlüsse verfügen, können nur vier verschiedene Drehregler abgefragt werden. (An jeden Joystickanschluß können zwei Drehregler angeschlossen werden.) Gibt man dann ein Argument, das größer als drei ist an, wird von der Reglernummer vier subtrahiert. Daher können Programme, die Eingaben von den Reglern 4 bis 7 erwarten, über die Regler 0 bis 3 gesteuert werden.
- Neben den Drehreglern können über PADDLE auch die Zeichentabletts „Atari Touch Tablet“ und „Koala-Pad“ abgefragt werden.

PEEK

Form: PEEK (nausdr)
nausdr: Adresse der abzufragenden Speicherzelle
Beispiel: LM=PEEK (82)

Die Funktion PEEK erlaubt es, den Inhalt des durch das Argument angegebenen Bytes abzufragen. Da der Mikroprozessor nur die Adressen von 0 bis 65535 kennt, sind Argumente auch nur in diesem Bereich sinnvoll.

Weitere Informatioanen dazu unter RAM, ROM, POKE, Speicheraufteilung, DPOKE und DPEEK.

```
10 REM ---------------------
20 REM Demonstration zu PEEK
30 REM ---------------------
40 REM linken Bildschirmrand zurueck-
50 REM setzen
60 POKE 82,0:GRAPHICS 0
70 REM Zeichenkette fuer Hexadezimal-
80 REM rechnung
90 DIM HEX$(16):HEX$="0123456789ABCDEF
"
100 REM Adressen eingeben
110 PRINT "Anfangsadresse ";
120 INPUT ANFANG
130 PRINT "Endadresse ";
140 INPUT SCHLUSS
150 REM Tabelle ausgeben
160 FOR I=ANFANG TO SCHLUSS:IN=PEEK(I)

170 PRINT I,IN,
180 HI=INT(IN/16):LO=IN-16*HI
190 PRINT HEX$(HI+1,HI+1);HEX$(LO+1,LO
+1),
200 POKE 766,1:PRINT CHR$(IN):POKE 766
,0
210 NEXT I
220 RUN
```

Abb. P.1: PEEK – Beispielprogramm

```
Anfangsadresse ?9626
Endadresse ?9644
9626      72      48      H
9627      105     69      i
9628      101     65      e
9629      114     72      r
9630      32      20
9631      115     73      s
9632      116     74      t
9633      101     65      e
9634      104     68      h
9635      101     65      e
9636      110     6E      n
9637      32      20
9638      68      44      D
9639      97      61      a
9640      116
```

Abb. P.2: PEEK – Beispiellauf

PEN (BASIC XL)

Form: PEN (nausdr)
nausdr: 0 – X-Position
1 – Y-Position
Beispiel: PLOT PEN(0),PEN(1)

Mit dieser Funktion kann die Position des Lichtgriffels (in Deutschland zur Zeit noch nicht erhältlich) festgestellt werden.

PLOT

Abkürzung: PL.
Form: PLOT nausdr1,nausdr2
nausdr1: Spalte
nausdr2: Zeile
Beispiel: PLOT X,Y

Durch PLOT wird an der angegebenen Bildschirmposition ein Punkt in einer bestimmten Farbe bzw. ein Zeichen ausgegeben. Die Farbe bzw. der ATASCII-Wert des Zeichens muß vorher durch COLOR festgesetzt worden sein.

Je nach Graphikbetriebsart ist die größtmögliche Anzahl von Spalten und Zeilen verschieden. Wird dieser veränderliche Höchstwert überschritten, kommt es zum Fehler 141 (Cursor out of range).

- Der PLOT-Befehl arbeitet intern über den Ein- und Ausgabekanal 7. Da dieser Kanal im normalen Textmodus nicht benutzt wird, muß man statt des Befehls GRAPHICS 0 das Kommando OPEN #6,12,0,"S:" verwenden, um PLOT gebrauchen zu können.

- Weitere Informationen hierzu unter COLOR und GRAPHICS.

PLOT...TO...TO... (MSB)

Form: PLOT nausdr1,nausdr2 (TO nausdr3, nausdr4 (TO...))
nausdr1/2: Koordinaten des ersten Punktes
nausdr3/4 und folgende: Koordinaten, zu denen Linien gezogen werden sollen
Beispiel: PLOT 0,0 TO 0,10 TO 10,10 TO 10,0 TO 0,0

Beim Microsoft-BASIC-Befehl PLOT können zusätzlich durch Anhängen von TO von dem gesetzten Punkt aus Linien gezeichnet werden. Der Beispielbefehl würde z. B. einen kompletten Rahmen zeichnen.

PMADR (BASIC XL)

Form: PMADR (nausdr)
nausdr: Nummer des Players
Beispiel: PLOADR=PMADR(0)

Mit Hilfe dieser Funktion kann der Anfang der Daten des angegebenen Objekts im Speicher ermittelt werden.

PMCLR (BASIC XL)

Abkürzung: PMCL.
Form: PMCLR (nausdr)
nausdr: Nummer des Players
Beispiel: PMCLR(PL)

Löscht den durch das einzige Argument angegebenen Player.

PMCOLOR (BASIC XL)

Abkürzung: PMC.
Form: PMCOLOR nausdr1,nausdr2,nausdr3
nausdr1: Nummer des Players
nausdr2: Farbton
nausdr3: Helligkeit
Beispiel: PMCOLOR 1,4,10

Mit diesem Kommando kann die Farbe eines Players bzw. Missiles festgesetzt werden.

PMGRAPHICS (BASIC XL)

Abkürzung: PMG.
Form: PMGRAPHICS nausdr
Beispiel: PMGRAPHICS 1

Dieses Kommando schaltet die Player-Missile-Graphik ein oder aus. Es gibt drei verschiedene mögliche Parameter:

0: Ausschalten der Player-Missile-Graphik
1: PM-Graphik mit halber vertikaler Auflösung (entspricht GRAPHICS 7)
2: PM-Graphik mit voller Auflösung (entspricht GRAPHICS 15)

PMMOVE (BASIC XL)

Abkürzung: PMM.
Form: PMMOVE nausdr1,nausdr2 (;nausdr3)
nausdr1: Nummer des Players
nausdr2: horizontale Position
nausdr3: Vektor für die vertikale Verschiebung
Beispiel: PMM. 0,100;−1

Mit diesem Befehl kann ein Objekt über den Bildschirm bewegt wer-

den. Dabei muß beachtet werden, daß die horizontale Position als Absolutwert, die vertikale jedoch als relativer Verschiebungswert angegeben werden muß. Im obigen Beispiel würde Objekt 0 auf die horizontale Position 100 gesetzt und gleichzeitig um eine Zeile nach oben verschoben werden.

PMWIDTH (BASIC XL)

Abkürzung: PMW.
Form: PMWIDTH nausdr1,nausdr2
nausdr1: Nummer des Players
nausdr2: Breite
Beispiel: PMW. 2,4

Mit diesem Befehl kann die Breite eines Objekts festgelegt werden. Dafür stehen die Werte 1 (normal), 2 (doppelte Breite) und 4 (vierfache Breite) zur Verfügung.

POKE

Abkürzung: POK.
Form: POKE nausdr1,nausdr2
nausdr1: Adresse
nausdr2: neuer Inhalt
Beispiel: POKE 710,0

Mit dem Befehl POKE kann einem bestimmten Byte im Speicher ein neuer Wert zugeordnet werden (sofern es sich dabei um RAM handelt).

- Bevor man POKE-Befehle ausprobiert, sollte man unbedingt das im Speicher befindliche BASIC-Programm abspeichern, da es passieren könnte, daß der Rechner abstürzt, so daß das Programm verloren geht.
- Lesen Sie zu diesem Thema auch den Abschnitt „Speicheraufteilung“.

POP (Atari BASIC, BASIC XL)

Abkürzung: nicht möglich
Form: POP
Beispiel: POP

Dieser Befehl muß immer dann benutzt werden, wenn man eine FOR-NEXT-Schleife ohne NEXT oder ein Unterprogramm ohne die Ausführung eines RETURN-Befehls verlassen möchte.

Auf diese Art und Weise wird verhindert, daß der Rechner die Zeilennummer, die er sich für den Rücksprung „gemerkt" hat, an anderer Stelle benutzt oder die Zeilennummer unnötig den internen Speicher für Rücksprungadressen belegt.

Hier ein Beispiel:

```
100 GOSUB 1000
.
.
1000 irgendwelche Befehle
.
1100 IF...THEN 900
1110 RETURN
```

So ist es falsch. Wenn eine solche Programmstruktur häufiger durchlaufen würde, wäre sehr bald nicht mehr ausreichend Speicherplatz vorhanden, um weitere Rücksprungadressen zwischenspeichern zu können.

So ist es richtig:

```
1100 IF...THEN POP:GOTO 900
```

Noch besser ist es allerdings, wenn man seine Programme so strukturiert, daß dieser „Notbehelf" unnötig ist.

POSITION (Atari, BASIC XL)

Abkürzung: POS.
Form: POSITION nausdr1,nausdr2
nausdr1: Spalte
nausdr2: Reihe
Beispiel: POSITION 2,2

Basic besitzt interne Speicherzellen, mit deren Hilfe es sich merkt, an welcher Stelle auf dem Bildschirm das nächste Zeichen ausgegeben werden wird. Das Kommando POSITION ermöglicht es, die Koordinaten dieses Punktes neu anzugeben und damit zu bestimmen, an welcher Bildschirmposition das nächste Zeichen ausgegeben wird.

Da durch den Befehl POSITION eben nur dieses interne Register verändert wird, bewegt sich der Cursor erst dann zur neuen Bildschirmposition, wenn ein Bildschirmausgabebefehl durchgeführt wird. Aus dem gleichen Grund werden Fehler, z. B. durch Setzen der Koordinaten auf eine außerhalb des Bildes liegende Position, erst bei der Ausführung des nächsten Bildschirmausgabebefehls (wie PRINT, PUT oder INPUT) auftreten.

PRINT

Abkürzung: PR. oder ?
Form: PRINT (#nausdr)(,)(;)(zausdr)(nausdr)
Beispiel: PRINT
?#6;23*Q
PR.#2;DAT$

Der Befehl PRINT ist ein vielseitiger Befehl zur Ausgabe von Zeichen in ATASCII-Form auf ein beliebigtes Peripheriegerät. Wenn man ein anderes Medium als den Bildschirm benutzen will, dann muß gleich nach dem Befehl die Nummer des betreffenden Ein- und Ausgabekanals angegeben werden. Dieser Kanal muß natürlich vorher für die Ausgabe auf das betreffende Gerät geöffnet worden sein.

Alle weiteren Argumente, die voneinander durch Kommas oder Semikolons getrennt sein müssen, werden auf das gewünschte Gerät ausgegeben.

Dabei wird wie folgt vorgegangen: Zeichenketten werden unverändert übertragen, numerische Ausdrücke werden berechnet, in ATASCII-Zeichen verwandelt und dann ausgegeben.

Das Komma und das Semikolon sind spezielle Steuerzeichen für die Ausgabe. Wenn zwei Teile des auszugebenden Textes durch ein Semikolon getrennt sind, wird bei der Ausgabe kein Zwischenraum erzeugt. Ein Komma dagegen bewirkt, daß zwischen den beiden Teilen so viele Leerzeichen eingesetzt werden, daß der Beginn der nächsten Spalte erreicht wird. Gibt man hinter dem letzten Textteil in der PRINT-Anweisung überhaupt kein Zeichen an, wird automatisch ein EOL-Zeichen eingesetzt, was auf dem Bildschirm und dem Drucker das Ende einer Zeile signalisiert und auf Diskette und Kassette einen einzelnen Eintrag in einer Datei (der dann wieder mit INPUT gelesen werden kann) beendet.

PRINT...AT (MSB)

Form: PRINT #nausdr,AT(nausdr,nausdr)...
Beispiel: PRINT #6,AT (3,3)"HALLO"

Mit dem Befehl PRINT...AT können Daten an einer bestimmten Position in der Datei ausgegeben werden (siehe auch INPUT...AT).

PRINT...USING (BASIC XL, MSB)

Diese Anweisung erlaubt es, Zahlen und Zeichenketten formatiert auszugeben. Dabei können die Anzahl der Stellen vor und nach dem Komma, die Vorzeichen, die Notation, die Bündigkeit und vieles mehr festgelegt werden.

PROTECT (BASIC XL)

Abkürzung: PRO.
Form: PROTECT zausdr
zausdr: Dateispezifikation einer Diskettendatei
Beispiel: PROTECT "D:*.BXL"

Dieser Befehl dient zum Schutz von Diskettendateien gegen versehentliches Umbenennen, Überschreiben oder Löschen.

- Wieder abschalten kann man diese Sicherung durch den Befehl UNPROTECT.
- Diese Art von Datensicherung hat nichts mit dem sogenannten „Schreibschutz" einer Diskette (durch Überkleben des Einschnitts am Diskettenrand) zu tun und bietet daher auch keinen Schutz gegen versehentliches Formatieren der Diskette.
- Wie fast alle anderen Diskettenfunktionen (im DOS-Menü: LOCK) läßt sich auch dieser Befehl durch ein XIO-Kommando simulieren. Im Microsoft-BASIC gibt es einen gleichbedeutenden Befehl namens LOCK.

PTRIG

Form: PTRIG (nausdr)
Beispiel: FEUER=PTRIG(2)

Die Funktion PTRIG ermittelt den Zustand des Feuerknopfs des angegebenen Drehreglers. Das Ergebnis ist dann Null, wenn der Knopf gedrückt ist.

- Genau wie auch bei den anderen Funktionen zur Abfrage der Joystickanschlüsse werden auf XL-Geräten Funktionsargumente zwischen 4 und 7 in 0 bis 3 umgewandelt.
- Mit der Funktion PTRIG können außerdem die Ausführungsknöpfe auf den Zeichentabletts „Atari Touch Tablet" und „KoalaPad" abgefragt werden.

PUT

Abkürzung: PU.
Form: PUT #nausdr1,nausdr2
nausdr1: Kanalnummer
nausdr2: auszugebender Wert
Beispiel: PUT#2,WERT

Mit dem Befehl PUT kann man auf jedes beliebige Peripheriegerät, das zur Ausgabe von Daten geeignet ist (also beispielsweise nicht die Tastatur), einzelne Bytes ausgeben. Dazu gibt man als erstes Argument die Nummer des Ausgabekanals und als zweites einen Wert zwischen 0 und 255 an.



RAD (Atari BASIC, BASIC XL)

Abkürzung: RA.
Form: RAD
Beispiel: RAD

Mit dem Befehl RAD wird in den Bogenmaßmodus geschaltet, wonach alle Argumente von trigonometrischen Funktionen als Bogenmaße interpretiert werden. Dieser Modus ist die Standardeinstellung; der Befehl; RAD muß nur dann benutzt werden, wenn aus dem Grad-Modus (siehe DEG) zurückgeschaltet werden soll.

RAM

RAM ist die Abkürzung für Random Access Memory (in etwa: Speicher mit wahlfreiem Zugriff). Diese Art von Speicher kann gelesen (PEEK) und beschrieben (POKE) werden; außerdem können alle Speicherstellen ohne weiteres und praktisch ohne Zugriffszeit angesprochen werden.

```
10 REM ------------------------
20 REM Demonstration zu RAM/ROM
30 REM ------------------------
35 POKE 82,0:PRINT :REM 40 Zeichen/Zei
le
40 PRINT "Welches Byte soll getestet w
erden";
50 INPUT BYTE
60 REM bisherigen Wert sichern
70 ALTERWERT=PEEK(BYTE)
80 REM neuen Wert berechnen
90 NEUER=ALTERWERT+1:IF NEUER>255 THEN
 NEUER=0
```

Abb. R.1: RAM – Beispielprogramm

```
100 REM neuen Wert ablegen
110 POKE BYTE,NEUER
120 REM Byte nochmals abfragen
130 INHALT=PEEK(BYTE)
140 IF INHALT=ALTERWERT THEN PRINT "RO
M oder Hardwareregister":GOTO 240
150 IF INHALT=NEUER THEN PRINT "RAM":G
OTO 240
160 PRINT "Speicherart nicht feststell
bar."
170 PRINT "Es koennte sich um"
180 PRINT "1.) Hardware-RAM"
190 PRINT "2.) nicht existierenden Spe
icher"
200 PRINT "oder um":PRINT "3.) RAM han
deln, dessen Inhalt staendig"
210 PRINT "veraendert wird."
220 GOTO 40
230 REM alten Wert zurueckschreiben
240 POKE BYTE,ALTERWERT
250 GOTO 40
```

Abb. R.1: RAM – Beispielprogramm (Forts.)

```
Welches Byte soll getestet werden?49152
Speicherart nicht feststellbar.
Es koennte sich um
1.) Hardware-RAM
2.) nicht existierenden Speicher
oder um
3.) RAM handeln, dessen Inhalt staendig
veraendert wird.
Welches Byte soll getestet werden?65535
ROM oder Hardwareregister
Welches Byte soll getestet werden?1536
RAM
Welches Byte soll getestet werden?48000
ROM oder Hardwareregister
Welches Byte soll getestet werden?
```

Abb. R.2: RAM – Beispiellauf

RANDOM (BASIC XL)

Form: RANDOM (nausdr)
Beispiel: WUERFEL=1+INT(RANDOM(6))

RANDOM dient zur Ermittlung einer Zufallszahl zwischen 0 und der als Parameter angegebenen Zahl. Das obenstehende Beispiel würde eine ganze Zufallszahl zwischen 1 und 6 berechnen.

RANDOMIZE (MSB)

Form: RANDOMIZE (nausdr)
Beispiel: RANDOMIZE 20

Da die Funktion RND im Microsoft-BASIC anders arbeitet als in den beiden anderen BASIC-Varianten, muß dem Programm der Befehl RANDOMIZE vorangestellt werden, damit nicht immer die gleichen Zufallszahlen produziert werden.

READ

Abkürzung: REA.
Form: READ varname(,varname...)
Beispiel: READ L,L$

Das Kommando READ dient dazu, aus den hinter DATA angegebenen Konstanten Werte in bestimmte Variablen zu übertragen. Dabei können Zahlenwerte sowohl in numerische als auch in Zeichenkettenvariablen eingelesen werden, während alphanumerische Daten nur in Zeichenkettenvariablen übertragen werden können.

Nach einem READ-Befehl wird ein interner Zeiger auf das nächste Element in der DATA-Zeile (oder wenn das Ende dieser Zeile erreicht war, auf das erste Element der nächsten Zeile) gesetzt.

Die Stellung dieses Zeigers kann mit Hilfe des Kommandos RESTORE beeinflußt werden, so daß man bestimmte Daten mehrfach lesen kann oder gezielt auf bestimmte Daten zugreifen kann.

Informationen über die Art der Daten, die Sie in DATA-Zeilen ablegen können, finden Sie unter DATA.

REM

Abkürzung: .
Form: REM beliebiger Text
Beispiel: REM *** Abspeicherung ***

Der Befehl REM hat überhaupt keine Auswirkung auf den Pro-

grammablauf und dient dazu, Kommentare in den Programmtext zu integrieren. Da nach REM alle Zeichen folgen dürfen, ist es nicht möglich, auf REM in der gleichen Zeile weitere Befehle folgen zu lassen.

REM-Befehle fördern die Lesbarkeit und die Verständlichkeit von Programmen und sollten deshalb zumindest an den Anfängen aller wichtigen Programmabschnitte auftauchen.

Obwohl der REM-Befehl keine programmtechnische Funktion hat, verbraucht auch er Speicherplatz. Deshalb ist es wichtig, als Sprungziel nie die Zeilennummer einer REM-Zeile anzugeben, damit man sie später löschen kann, ohne die Funktionsfähigkeit des Programms zu beeinträchtigen.

Ein Beispiel: statt

```
100 GOSUB 2000
.
.
.
2000 REM Initialisierung
20010 ...
```

sollte man

```
19999 REM Initialisierung
20000 ...
```

schreiben.

RENAME (BASIC XL)

Form: zausdr
zausdr: Alter Dateiname, neuer Dateiname (durch Komma getrennt)
Beispiel: RENAME "D1:*.CAR,*.BAS"

Mit RENAME können Diskettendateien umbenannt werden. Beachten Sie dabei, daß die beiden Dateinamen innerhalb einer Zeichenkette, voneinander durch ein Komma getrennt, stehen müssen.

Beim zweiten Dateinamen darf nur der eigentliche Name, nicht aber die Gerätespezifikation angegeben werden, da BASIC XL davon ausgeht, daß sich die umzubenennenden Dateien nach der Umbenennung noch auf der gleichen Diskette befinden werden.

Im Atari-BASIC läßt sich diese Funktion mit einem X10-Kommando simulieren, in Microsoft-BASIC steht hierfür der Befehl NAME zur Verfügung.

RENUM (BASIC XL)

Form: RENUM (nausdr1) (,nausdr2)
nausdr1: neue Anfangszeile
nausdr2: Schrittweite
Beispiel: RENUM
RENUM 100
RENUM ,5
RENUM 1000,10

RENUM ermöglicht es, die Zeilennummern eines BASIC-Programms neu durchzunumerieren. Dabei werden auch die Zeilennummern hinter GOTO, GOSUB, RESTORE etc. so verändert, daß die Zeilennummern wieder übereinstimmen.

Als Argument können die neue Anfangszeilennummer und die Schrittweite angegeben werden.

Vorsicht: RENUM ist nicht dazu in der Lage, Programme richtig zu numerieren, in denen Zeilennummern keine Konstanten sind (Beispiel: RESTORE 10000+100*I)!

RENUM (MSB)

Form: RENUM nausdr1,nausdr2,nausdr3
nausdr1: Zeile, ab der umnumeriert werden soll
nausdr2: neue Anfangszeile
nausdr3: Schrittweite
Beispiel: RENUM 10,10,10

Der Befehl RENUM numeriert die Programmzeilen von Zeile nausdr1 ab um, wobei die neue Startzeilennummer nausdr2 und die Schrittweite nausdr3 ist.

RESTORE

Abkürzung: RES.
Form: RESTORE (nausdr)
nausdr: Zeilennummer
Beispiel: RES.
RESTORE 20010

BASIC verfügt über einen internen Zeiger, der jeweils auf das Element einer DATA-Zeile zeigt, das beim nächsten READ-Befehl gelesen werden wird. Mit dem Befehl RESTORE kann man den Zeiger auf eine bestimmte DATA-Zeile (RESTORE Zeilennummer) oder auf die erste DATA-Zeile im Programm (RESTORE ohne Zeilennummer) setzen. Auf diese Art und Weise ist es möglich, mit READ entweder auf bestimmte Datenfelder oder auf ein Datenfeld mehrfach zuzugreifen.

RESUME (MSB)

Form: RESUME
RESUME NEXT
RESUME nausdr
Beispiel: RESUME 343

Nach Bearbeitung eines „ON ERROR"-Befehls kann das Programm mit RESUME fortgesetzt werden. Dabei sind vier verschiedene Möglichkeiten vorhanden: RESUME setzt das Programm bei dem Befehl fort, wo der Fehler aufgetreten war, „RESUME NEXT" bei dem darauffolgenden Befehl und „RESUME zeilennummer" bei der angegebenen Programmzeile fort.

RETURN

Abkürzung: RET.
Form: RETURN
Beispiel: RETURN

Am Ende jedes Unterprogramms, das mit GOSUB angesprungen worden ist, muß ein RETURN-Befehl stehen. Wenn die Programmausführung zu diesem Kommando gelangt, wird zu dem Befehl verzweigt, der hinter dem GOSUB-Befehl, der das Unterprogramm aufgerufen hatte, steht.

Ein Fehler kann dann auftreten, wenn vor Ausführung von RETURN kein GOSUB erfolgt war.

Weitere Informationen hierzu unter POP und GOSUB.

RGET (BASIC XL)

Abkürzung: RG.
Form: RGET #nausdr1,varname (,varname,...)
nausdr: Kanalnummer
varname: Namen der Variablen, in die Werte eingelesen werden sollen
Beispiel: RGET #2,A,A$

Mit RGET können die Werte von Variablen, die mit RPUT abgespeichert worden sind, wieder in Variablen eingelesen werden. Dabei muß allerdings darauf geachtet werden, daß nicht alphanumerische und numerische Werte vertauscht werden. Außerdem müssen Sie beachten, daß eine Zeichenkettenvariable, in die ein Wert übertragen werden soll, vorher auf die gleiche Länge dimensioniert werden muß wie die ursprüngliche Variable, die abgespeichert worden war.

RIGHT$ (BASIC XL, MSB)

Form: RIGHT$(zausdr,nausdr)
zausdr: Zeichenkette
nausdr: Anzahl der Zeichen
Beispiel: RIGHT$("Handbuch",4)

Mit RIGHT$ kann man die letzten Zeichen einer Zeichenkette „entnehmen". Im obigen Beispiel würden als Wert die letzten vier Zeichen, also die Zeichenkette „buch" bestimmt.

- Im Atari-BASIC ließe sich diese Funktion beispielsweise folgendermaßen darstellen:

```
DAT$="Handbuch"
L=LEN(DAT$
?DAT$(L-3,L)
```

Man sieht, daß hier die Nachahmung dieser Funktion einen gewissen Programmieraufwand (im Gegensatz zu LEFT$) verlangt.

ROM

ROM ist die Abkürzung für Read Only Memory (Speicher, der nur gelesen werden kann).

ROM-Speicher werden immer dann verwendet, wenn die jeweiligen Daten ohne Probleme sofort nach dem Einschalten des Computers verfügbar sein sollen. Beim Atari befinden sich BASIC, Betriebssystem und Selbsttestprogramm im ROM-Speicherbereich und sind somit gleich nach Anschalten des Computers verfügbar.

RND (Atari BASIC, BASIC XL)

Form: RND(nausdr)
Beispiel: ? RND(0)

Die Funktion RND liefert unabhängig vom Funktionsargument, das somit eine sogenannte Dummyvariable ist, Zufallszahlen zwischen 0 und 1.

- In BASIC XL ist zusätzlich die Funktion RANDOM verfügbar.

RND (MSB)

Form: RND (nausdr)
nausdr: Obergrenze des Zahlenbereichs
Beispiel: RND(100)

RND funktioniert genauso wie im Atari-BASIC. Zusätzlich kann sie jedoch auch ohne Parameter benutzt werden (der Bereich liegt dann

```
10 REM --------------------
20 REM Demonstration zu RND
30 REM --------------------
40 REM Variablen initialisieren
50 DIM ZAHLEN$(49),ERG(7)
60 FOR I=1 TO 49
70 ZAHLEN$(I,I)="1"
80 NEXT I
90 REM Zufallszahlen berechnen
100 FOR I=1 TO 7
110 ZUFALL=1+INT(49*RND(0))
120 IF ZAHLEN$(ZUFALL,ZUFALL)<>"1" THE
N 110:REM Schon mal dagewesen
130 ZAHLEN$(ZUFALL,ZUFALL)="0":REM Als
 benutzt kennzeichnen
140 ERG(I)=ZUFALL
150 NEXT I
160 REM Die ersten 6 Zahlen sortieren
170 FOR I=1 TO 6
180 FOR J=I TO 6
190 IF ERG(I)>ERG(J) THEN ZWISCHEN=ERG
(J):ERG(J)=ERG(I):ERG(I)=ZWISCHEN
200 NEXT J
210 NEXT I
220 REM Ausgeben
230 PRINT "Die Lottozahlen der Woche:"

240 FOR I=1 TO 6
250 PRINT ERG(I);" ";
260 NEXT I
270 PRINT :PRINT "Zusatzzahl: ";ERG(7)

280 PRINT :PRINT "Ohne Gewaehr!"
```

Abb. R.3: RND – Beispielprogramm

```
Die Lottozahlen der Woche:
15 22 29 39 45 49
Zusatzzahl: 10

Ohne Gewaehr!

READY
RUN
Die Lottozahlen der Woche:
14 18 20 38 45 47
Zusatzzahl: 46

Ohne Gewaehr!
```

Abb. R.4: RND – Beispiellauf

zwischen 0 und 1) oder Zufallszahlen in einem bestimmten Zahlenbereich erzeugen (im obigen Beispiel zwischen 0 und 100).

- Diese erweiterte Random-Funktion bietet ebenfalls BASIC XL durch RANDOM.

Routine

Als Routine bezeichnet man gewöhnlich einen bestimmten Teilabschnitt eines Programms oder eines Unterprogramms, das eine fest umrissene Aufgabe zu erfüllen hat.

RPUT (BASIC XL)

Abkürzung: RP.
Form: RPUT #nausdr,varname(,varname,...)
nausdr: Kanalnummer
varname: Namen der Variablen, deren Werte abgespeichert werden sollen
Beispiel: RPUT #2,A,A$

Mit RPUT können die Inhalte von Variablen über irgendeinen I/O-Kanal abgespeichert werden.

Lesen Sie hierzu auch die Bemerkungen unter RGET!

RUN

Abkürzung: RU.
Form: RUN (zausdr)
zausdr: Name des Programms (mit Gerätespezifikation)
Beispiel: RUN
RUN "C:"
RUN "D:WEITER.BAS"

Der Befehl RUN kann auf zwei verschiedene Arten verwendet werden. Ohne weitere Argumente startet er das im Speicher befindliche Programm. Man kann aber auch als Argument eine Gerätespezifikation angeben. Das Programm wird dann vom angegebenen Gerät geladen und automatisch gestartet. Es wird dabei so vorgegangen, als wenn LOAD zausdr und danach RUN angegeben worden wäre.



SAVE

Abkürzung: SA.
Form: SAVE zausdr
zausdr: Dateispezifikation
Beispiel: SAVE "C:"

Der Befehl SAVE ermöglicht es, im Speicher befindliche BASIC-Programme abzuspeichern. Dazu gibt man hinter SAVE den Gerätenamen und bei Diskettenstation auch einen Dateinamen an.

Bei der Arbeit mit Kassettenrecorder sind die gleichen Hinweise zu beachten, wie bei CSAVE.

SAVE...LOCK (MSB)

Form: SAVE zausdr LOCK
zausdr: Gerätespezifikation
Beispiel: SAVE "D:TEST.MSB" LOCK

Wird in Microsoft-BASIC bei SAVE hinter den Programmnamen der Begriff LOCK gesetzt, dann wird das abzuspeichernde Programm gleichzeitig so verändert, daß es sich nicht mehr listen oder verändern läßt. (Es ist allerdings äußerst fraglich, ob ein derartiger Programmschutz ausreicht, um das Programm wirklich sicher zu schützen – jeder Schutz, der auf irgendeine Art und Weise von einem Programm erzeugt wird, läßt sich auch beseitigen.) Denken Sie daran, daß Sie sich für den privaten Gebrauch eine ungeschützte Kopie aufbewahren sollten.

SCRN$ (MSB)

Form: SCRN$(nausdr1,nausdr2)
nausdr1: Spalte
nausdr2: Zeile
Beispiel: SCRN$ (6,9)

Mit der Funktion SCRN$ läßt sich bestimmen, welches Zeichen an einer bestimmten Position auf dem Bildschirm steht (siehe Atari-BASIC-Befehl LOCATE).

SET (BASIC XL)

Abkürzung: nicht möglich
Form: SET nausdr1,nausdr2
nausdr1: Nummer der OPTION
nausdr2: neuer Wert
Beispiel: SET 1,5

Mittels des Kommandos SET können verschiedene Voreinstellungen im BASIC XL verändert werden. Dazu gibt man einfach als Werte nur die Nummer der Option und den neuen Wert an.

Es folgt eine Liste der veränderbaren Einstellungen und der möglichen Werte:

Option	Werte	Bedeutung
0	0	BREAK funktioniert wie gewohnt (Normaleinstellung).
	1	Drücken von BREAK führt zu Fehlermeldung.
	128	BREAK wird ignoriert.
1	1–128	Breite der Spalte, die mit Hilfe des Kommas bei PRINT erzeugt werden (Normalstellung: 10).
2	0–255	ATASCII-Wert des Zeichens, das bei INPUT automatisch erzeugt wird: Normalwert ist 63, der ATASCII-Wert des Fragezeichens.
3	0	FOR-NEXT-Schleifen werden mindestens einmal durchlaufen (Normalstellung, wie in Atari-BASIC).

	1	Es kann vorkommen, daß FOR-NEXT-Schleifen keinmal durchlaufen werden (internationaler Standard).
4	0	Bei Eingabe von zu wenig Werten bei INPUT wird mit „??“ nach den restlichen Werten gefragt (Normalstellung).
	1	Statt dessen wird eine Fehlermeldung erzeugt.
5	0	Bei der Programmeingabe darf auch Klein- und Inversschrift benutzt werden (Normaleinstellung).
	1	Alle Zeichen (außer jenen, die zwischen Anführungsstrichen stehen) werden in normale Großschrift verwandelt (Atari-BASIC-Standard).
6	0	Zusätzlich zu den Fehlernummern werden Texte ausgegeben (Normaleinstellung).
	1	Es werden nur die Fehlermeldungen ausgegeben.
7	0	Missiles, die vertikal aus dem Bild verschwinden, gehen verloren,
	1	tauchen auf der gegenüberliegenden Seite wieder auf.
8	0	Bei USR wird die Anzahl der Parameter nicht auf den Stack übertragen.
	1	USR funktioniert wie in Atari-BASIC (Normaleinstellung).
9	0	Nach ENTER kehrt das BASIC in den Eingabemodus zurück (Normaleinstellung).
	1	Wenn ein TRAP-Befehl vorher ausgeführt worden ist, wird die entsprechende Zeilennummer im Fall eines Fehlers angesprungen.
10	0	Die vier Missiles arbeiten wie vier voneinander getrennte Objekte (Normaleinstellung).
	1	Die Missiles bilden einen fünften Player.
11	1–255	Ein String, der nicht vorher dimensioniert worden ist, wird im Falle seines erstmaligen Auftretens automatisch auf die vorgegebene Länge dimensioniert (Voreinstellung: 40).
12	0	Das Listing wird wie in Atari-BASIC ausgegeben.
	1	Bei LIST werden FOR-NEXT-, IF-THEN-ELSE- und WHILE-ENDWHILE-Schleifen eingerückt ausgegeben (Normaleinstellung).

SETCOLOR (Atari, BASIC XL)

Abkürzung: SE.
Form: SETCOLOR nausdr1,nausdr2,nausdr3
nausdr1: Nummer des Farbregisters
nausdr2: Farbe
nausdr3: Helligkeit
Beispiel: SETCOLOR 2,0,0

Der Atari verfügt über neun Speicherstellen, in denen er bestimmte Farbwerte abspeichert. Die ersten vier „Farbregister" (704–707) sind für die vier Player-Missile-Objekte reserviert. Die anderen fünf (708–712) können mit Hilfe des SETCOLOR-Befehls verändert werden. Dazu gibt man als erstes Argument die Nummer des Farbregisters (0–4), als zweites die Nummer der Farbe (siehe Tabelle unter „Farben") und als letztes die Helligkeit an (gerade Zahlen zwischen 0 und 15). Nur in den Graphikstufen 9 bis 11 gibt es tatsächlich 16 verschiedene Helligkeitsstufen jeder Farbe.

Je nach benutzter Graphikstufe steuern die Farbregister die Farben verschiedener Bildschirmteile. Genauere Informationen hierzu finden Sie unter „COLOR".

In manchen Graphikstufen und bei der Benutzung der Player-Missile-Graphik ist es nötig, den Wert eines der ersten vier Farbregister festzulegen. Dazu geht man folgendermaßen vor: Man multipliziert die Nummer der Farbe mit 16 und addiert dazu die Helligkeit. Diesen Wert überträgt man durch POKE in das entsprechende Farbregister.

SETCOLOR (MSB)

Der SETCOLOR-Befehl im Microsoft-BASIC unterscheidet sich in einem Punkt von dem „normalen" Befehl SETCOLOR: Da zusätzlich die Farbregister für die Player-Missile-Graphik zugänglich sind (SETCOLOR 0–3), sind die anderen Register verschoben, so daß SETCOLOR 0,... in Atari-BASIC dem SETCOLOR 4,... im Microsoft-BASIC entspricht.

SGN

Form: SGN(nausdr)
Beispiel: SGN(QW2)

Mit der Funktion SGN läßt sich das Vorzeichen eines numerischen Ausdrucks bestimmen. Das Ergebnis ist „−1“, wenn der Ausdruck einen negativen Wert hat (kleiner als Null ist), „0“, wenn der Wert Null ist und „1“, wenn der Ausdruck eine positive Zahl darstellt (größer als Null ist).

Eine Verwendungsmöglichkeit bietet sich dann, wenn Anfangs- und Endwert einer FOR-NEXT-Schleife nicht vorher festgelegt sind (weil sie etwa unter Anwendung von RND berechnet werden). In einem solchen Fall kann man als Schrittweite den Ausdruck „SGN“(ENDWERT-ANFANGSWERT) angeben. Die Schleife wird dann automatisch in der richtigen Richtung durchlaufen. Doch Vorsicht: Wenn beide Werte gleich sind, wird die Schleife zu einer endlosen Schleife, die dann nur durch eine Unterbrechung des laufenden Programms beendet werden kann.

SIN

Form: SIN(nausdr)
Beispiel: DEG:?SIN(90)

Mit der Funktion SIN kann man den Sinus eines Bogenmaßwertes (wenn Sie sich im RAD-Modus befinden) oder eines Winkels (im DEG-Modus) bestimmen.

SOUND (Atari, BASIC XL)

Abkürzung: SO.
Form: SOUND nausdr1,nausdr2,nausdr3,nausdr4
nausdr1: Tonkanal

nausdr2: Tonhöhe
nausdr3: Verzerrung
nausdr4: Lautstärke
Beispiel: SOUND 0,100,10,8
SOUND 2,0,0,0

Die vier Tongeneratoren des Atari können mit Hilfes des Kommandos SOUND angesprochen werden.

Der erste Parameter gibt dabei die Nummer des Tongenerators, die zwischen 0 und 3 liegen kann, an. Daraus folgt, daß zwar für jeden Tonkanal ein eigener Befehl gegeben werden muß, andererseits aber auch, daß jeder Kanal gleichwertig ist und gleiches leisten kann.

Mit Hilfe der zweiten Zahl wird die Tonhöhe festgelegt. Der Wert darf zwischen 0 und 255 liegen, wobei der Ton umso höher wird, je kleiner dieser Wert ist (exakt: Die Frequenz des Tons ist dem Quotienten aus Taktfrequenz und dem Tonhöhenwert proportional). Im folgenden eine Tabelle der Tonhöhenwerte für die Noten, die innerhalb des ansprechbaren Tonbereichs liegen.

Notenname	Tonhöhenwert	Notenname	Tonhöhenwert
C	29	F	91
H	31	E	96
A# / B	33	D# / Eb	102
A	35	D	108
G# / Ab	37	C# / Db	114
G	40	C	121
F# / Gb	42	H	128
F	45	A# / B	136
E	47	A	144
D# / Eb	50	G# / Ab	153
D	53	G	162
C# / Db	57	F# / Gb	173
C	60	F	182
H	64	E	193
A# / B	68	D# / Eb	204
A	72	D	217
G# / Ab	76	C# / Db	230
G	81	C	243
F# / Gb	85		

Man kann hier übrigens leicht feststellen, daß die Erhöhung des Tons um eine Oktave immer ungefähr einer Halbierung des Tonwertes entspricht.

Der dritte Wert wird benutzt, um die Verzerrung des Tons zu bestimmen. Hierbei sind nur die geraden Werte bis 14 sinnvoll. Gibt man als Argument eine 10 an, dann wird ein Sinuston, also ein vollkommen reiner Ton erzeugt. Alle anderen Werte mischen dem Ton ein mehr oder minder starkes Rauschen bei.

Das letzte Argument schließlich gibt die Lautstärke des Tons an. Sie darf zwischen 1 und 15 liegen. Allerdings sollte darauf geachtet werden, daß die Gesamtlautstärke der vier Tonkanäle nicht den Wert 32 überschreitet, da es sonst zu Verzerrungen kommen könnte. Der Tongenerator kann abgeschaltet werden, indem als Lautstärkewert 0 angegeben wird.

- Wird ein laufendes Programm mit BREAK unterbrochen, während Töne erzeugt werden, bleiben die Tongeneratoren in Betrieb. Um alle Tongeneratoren abzuschalten, können Sie einfach END eingeben (beachten Sie dabei aber auch die anderen Auswirkungen von END!).

SOUND (MSB)

Auch beim SOUND-Befehl gibt es im Microsoft-BASIC einen Unterschied: Zusätzlich zu den ersten vier Parametern kann als fünfter Wert die Länge des Tons in Jiffies (1/50 Sekunden) angegeben werden.

```
10 REM -----------------------
20 REM Demonstration zu SOUND
30 REM -----------------------
40 REM Bildschirm initialisieren
50 GRAPHICS 1+16
60 POSITION 0,0
70 PRINT #6;"       sound-tester"
80 POSITION 8,5:PRINT #6;"KANAL:0"
90 POSITION 8,8:PRINT #6;"HOEHE:0"
100 POSITION 3,11:PRINT #6;"VERZERRUNG
:0"
```

Abb. S.1: SOUND – Beispielprogramm

```
110 POSITION 2,14:PRINT #6;"LAUTSTAERK
E:0"
115 POSITION 2,18:PRINT #6;"knopf= LOE
SCHEN"
120 REM Variablen
130 CLR :OPT=1:KANAL=0:DIM HOEHE(3),VE
RZ(3),LAUT(3)
135 FOR I=0 TO 3:LAUT(I)=0:VERZ(I)=0:H
OEHE(I)=0:NEXT I
140 REM Wenn der Joystick nicht be-
150 REM wegt wird -> warten
160 IF STICK(0)=15 THEN 260
165 POSITION 0,OPT*3+2:PRINT #6;" ";
170 S=STICK(0)
180 OPT=OPT-(S=6)-(S=10)-(S=14)+(S=5)+
(S=9)+(S=13)
190 IF OPT>4 THEN OPT=1
200 IF OPT<1 THEN OPT=4
210 POSITION 0,OPT*3+2:PRINT #6;">";
220 H=PTRIG(0)-PTRIG(1)
230 ON OPT GOSUB 2000,2100,2200,2300
240 REM Ton einschalten
250 SOUND KANAL,HOEHE(KANAL),VERZ(KANA
L),LAUT(KANAL)
260 IF  NOT STRIG(0) THEN RUN
270 GOTO 160
1999 REM Kanal
2000 KANAL=KANAL+H
2010 IF KANAL<0 THEN KANAL=3
2020 IF KANAL>3 THEN KANAL=0
2030 POSITION 14,5:PRINT #6;KANAL
2032 POSITION 14,8:PRINT #6;HOEHE(KANA
L);"   "
2034 POSITION 14,11:PRINT #6;VERZ(KANA
L);" "
2036 POSITION 14,14:PRINT #6;LAUT(KANA
L);" "
2040 RETURN
2099 REM Tonhoehe
2100 HOEHE(KANAL)=HOEHE(KANAL)+H
2110 IF HOEHE(KANAL)<0 THEN HOEHE(KANA
L)=255
2120 IF HOEHE(KANAL)>255 THEN HOEHE(KA
NAL)=0
2130 POSITION 14,8:PRINT #6;HOEHE(KANA
L);"   "
2140 RETURN
2199 REM Verzerrung
2200 VERZ(KANAL)=VERZ(KANAL)+H*2
2210 IF VERZ(KANAL)>15 THEN VERZ(KANAL
)=0
2220 IF VERZ(KANAL)<0 THEN VERZ(KANAL)
=14
2230 POSITION 14,11:PRINT #6;VERZ(KANA
L);" "
2240 RETURN
2299 REM Lautstaerke
2300 LAUT(KANAL)=LAUT(KANAL)+H
2310 IF LAUT(KANAL)>15 THEN LAUT(KANAL
)=0
```

Abb. S.1: SOUND – Beispielprogramm (Forts.)

```
2320 IF LAUT(KANAL)<0 THEN LAUT(KANAL)
=15
2330 POSITION 14,14:PRINT #6;LAUT(KANA
L);" "
2340 RETURN
```

Abb. S.1: SOUND – Beispielprogramm (Forts.)

```
10 REM ----------------------
20 REM Demonstration zu Sound
30 REM ----------------------
40 REM Durch Veraendern des Laut-
50 REM staerkeverlaufs des Tones
60 REM lassen sich interessante
70 REM Effekte erzielen
80 FOR I=0 TO 240 STEP 10
90 FOR J=15 TO 7 STEP -1
100 SOUND 0,I+10,14,J
110 SOUND 1,I,14,J/2
120 NEXT J
130 NEXT I
```

Abb. S.2: SOUND – Beispielprogramm

Speicheraufteilung

Das Herz des Atari-Computers ist der Mikroprozessor 6502C (6502 in den älteren Geräten). Bei diesem handelt es sich um einen 8-Bit-Prozessor, das heißt, daß er mit Zahlen zwischen 0 und 255 (2^8-1) rechnen kann. Zur Adressierung von Speicherstellen stehen ihm aber 16 Bits zur Verfügung, so daß er Speicherstellen von 0 bis 65535 ($2^{16}-1$) kennt. (Sie werden im folgenden sehen, daß gerade bei der Arbeit mit Speicheradressen die Verwendung des hexadezimalen Zahlenformats Vorteile bietet.)

Ihr Computer enthält zunächst einmal zwei verschiedene Arten von Speicher: einerseits das RAM, dessen Inhalt verändert werden kann und beim Ausschalten des Geräts verlorengeht, andererseits das ROM, das sich nicht verändern läßt, dafür aber auch seinen Inhalt nicht verlieren kann. Aus diesem Grunde sind beispielsweise das

BASIC und das Betriebssystem im ROM abgespeichert. Außerdem benutzt der Atari einen bestimmten Adreßbereich als „Verbindung" zu seinen Hilfsprozessoren ANTIC, GTIA, POKEY und PIA. Versucht man, einer dieser Adressen einen Wert zuzuweisen (POKE), dann wird in dem betreffenden Baustein eine bestimmte Funktion ausgelöst. Liest man den Wert einer dieser Speicherstellen (PEEK), dann erhält man zumeist Informationen aus dem Prozessor.

65535 Bytes oder 64 K-Byte kann der Mikroprozessor also verwalten. Wie kann es aber sein, daß beispielsweise der Atari 800 XL 64 K-Byte RAM und zusätzlich BASIC und Betriebssystem verwalten kann? Die Antwort: Er kann es nicht! So sind von den 64 K-Byte RAM im 800 XL vom BASIC aus für den Prozessor nur 40 K-Byte sichtbar. Denn 8 K-Byte liegen unter dem Adreßbereich des BASIC und 16 K-Byte unter dem des Betriebssystems verborgen. Mit anderen Worten: Es ist nicht ohne weiteres möglich, diesen zusätzlichen Speicherbereich zu nutzen.

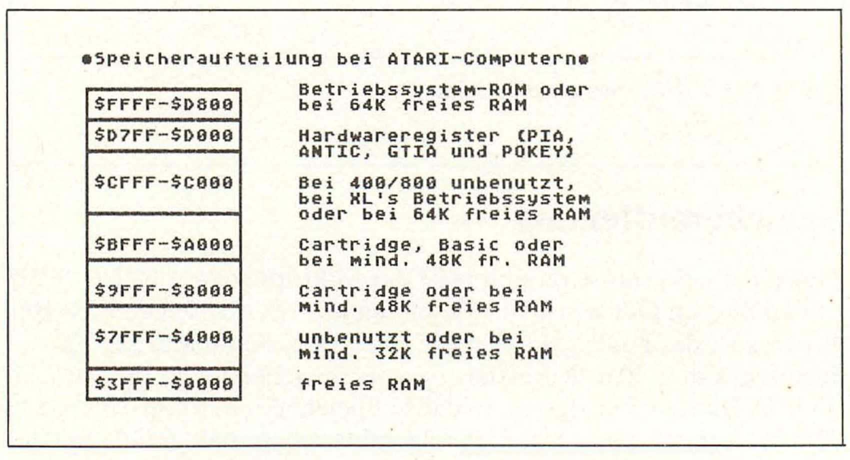


Abb. S.3: Speicheraufteilung

SQR

Form: SQR (nausdr)
Beispiel: ?SQR(9)

Die Funktion SQR dient dazu, die Quadratwurzel einer Zahl zu bestimmen.

Denken Sie daran, daß das Argument größer oder gleich Null sein muß!

STACK (MSB)

Form: STACK
Beispiel: IF STACK THEN SOUND 1,1,1,1,1 ELSE CLEAR STACK

Die Funktion STACK gibt an, für wie viele Befehle, die interne Zähler benötigen (z. B. SOUND, AFTER etc.), noch im internen Speicher Platz ist. Ist kein Platz mehr vorhanden, so kann man ihn durch CLEAR STACK löschen, was natürlich nicht während der Abarbeitung eines der betreffenden Befehle passieren darf.

STATUS

Abkürzung: ST.
Form: STATUS #nausdr,nvar
Beispiel: STARUS #1,ZUSTAND

Mit STATUS kann der Betriebszustand eines bestimmten Ein- und Ausgabekanals (erster Parameter) abgefragt werden. Das Resultat dieser Abfrage wird in der numerischen Variablen abgespeichert, deren Name als zweiter Parameter angegeben worden ist. (Die Fehlernummern entsprechen den unter ERROR angegebenen; 1 bedeutet, daß alles in Ordnung ist.)

STEP

Siehe unter FOR-NEXT.

STICK

Form: STICK (nausdr)
nausdr: Nummer des Joysticks
Beispiel: ?STICK(0)

Mit der Funktion STICK kann die Stellung der Steuerknüppel abgefragt werden. Da an dem Atari 400 und 800 vier verschiedene Steuerknüppel angeschlossen werden können, darf das Argument, das die Nummer des Joysticks angibt, zwischen 0 und 3 liegen. Das Problem, daß die neuen Modelle 600 XL und 800 XL nur noch zwei Anschlüsse besitzen, ist dadurch gelöst worden, daß hier STICK(2) und STICK(3) stets die gleichen Ergebnisse liefern wie STICK(0) und STICK(1). Diese Lösung bringt selbstverständlich nur bei den Programmen, in denen nicht alle Joysticks gleichzeitig benutzt werden können, Vorteile.

Als Ergebnis liefert die Funktion neun verschiedene Werte, die für den Laien eigentlich in ihrer Anordnung keinen Sinn ergeben können.

10 14 06
11 15 07
09 13 05

(Die Funktion hat also den Wert 15, wenn sich der Joystick in der Grundstellung befindet; eine Bewegung nach links würde beispielsweise den Wert 07 liefern.)

Durchschaubarer werden die Werte, wenn man sie binär wiedergibt:

1010 1110 0110
1011 1111 0111
1001 1101 0101

Man erkennt jetzt leicht, daß jeder der vier Hauptrichtungen des Joysticks ein Bit zugeordnet ist, das auf Null gesetzt wird, wenn der Steuerknüppel in die entsprechende Richtung bewegt wird. Drückt man den Knüppel schräg, dann werden eben zwei Bits gleichzeitig auf Null gesetzt, bewegt man ihn gar nicht, dann behalten alle vier Bits den Wert 1, und es entsteht das Resultat 15 ($8+4+2+1=2^3+2^2+2^1+2^0=15$).

Aufgrund dieses Aufbaus der Funktionswerte ist es im Prinzip sehr leicht möglich, auch diagonale Bewegungen richtig zu interpretieren, indem man den Status der einzelnen Bits überprüft. Hierfür gibt es leider im Atari-BASIC keinen Befehl (wie zum Beispiel in BASIC XL), so daß man sich anders behelfen muß.

Der folgende Ausdruck liefert eine 1, wenn der Joystick schräg oder gerade nach oben gedrückt wird und −1, wenn er gerade oder schräg nach unten gedrückt wird. Nur wenn er gar nicht oder genau horizontal bewegt wird, ist das Ergebnis dieses Terms 0.

S=STICK(0)
?−(S=6)−(S=10)−(S=14)+(S=5)+(S=9)+(S=13) (siehe auch unter „logische Ausdrücke").

Für den gleichen Zweck kann man sich für die Horizontale eines kleinen Tricks bedienen. Da für den Computer die Knöpfe an den Drehreglern mit den Schaltern für „rechts" und „linkgs" im Joystick identisch sind, kann man hier die folgende Formel benutzen (N sei die Nummer des Joysticks):

?PTRIG(2*N+1)−PTRIG(2*N)

STOP

Abkürzung: STO.
Form: STOP
Beispiel: STOP

Der Befehl STOP unterbricht das laufende Programm, ohne die Tonregister zurückzusetzen oder die Ein- und Ausgabekanäle zu schließen. Da jedoch die Meldung „STOPPED AT LINE Zeilennummer" ausgegeben wird und dazu in den normalen Textmodus zurückgewechselt wird, wird ein eventuell benutzter Graphikbildschirm gelöscht und abgeschaltet.

Die Programmausführung kann mit CONT fortgesetzt werden, wobei aber darauf geachtet werden muß, daß sich der Bildschirminhalt verändert haben wird.

STRIG

Form: STRIG(nausdr)
nausdr: Nummer des Steuerknüppels
Beispiel: IF NOT STRIG(SP) THEN PRINT "FEUER"

Die Funktion STRIG dient dazu, den Zustand der Feuerknöpfe an den Steuerknüppeln abzufragen. Das Argument kann zwischen 0 und 3 liegen, da man an den Atari 400 und 800 bis zu vier Steuerknüppel anschließen kann. Auf den XL-Modellen wird man für STRIG(2) und STRIG(3) die gleichen Ergebnisse bekommen wie für STRIG(0) und STRIG(1) (siehe dazu auch unter STICK).

Das Funktionsergebnis ist 0, wenn der Knopf gedrückt ist und eins, wenn er nicht gedrückt ist.

STRING$ (MSB)

Form: STRING$(numausdr,ausdr)
Beispiel: LEER$=STRING$(40," ")

Die Funktion STRING$ erzeugt eine Zeichenkette der durch „numausdr" angegebenen Länge, die entweder aus der durch die Zeichenkette oder durch den Zeichenwert angegebenen Zahl besteht.

STR$

Form: STR$(nauzzzsdr)
Beispiel: STR$(Q)
STR$(3*P+13)

Die Funktion STR$ bietet die Möglichkeit, eine Zahl in eine Zeichenkette zu verwandeln, die dann die einzelnen Ziffern in ATASCII-Form enthält.

Eine Rückumwandlung in eine Zahl ist durch die Funktion VAL möglich.

SYS (BASIC XL)

Form: SYS (nausdr)
nausdr: Nummer des internen Registers von BASIC XL
Beispiel: IF SYS(12)=0 THEN SET 12,0:?"Finden Sie denn nicht auch das Ausgabeformat von BASIC XL schoener?"

Die Funktion SYS erlaubt es, die internen Register, mit denen man bestimmte Funktionen von BASIC XL kontrollieren kann (siehe auch SET) auf ihren Status hin abzufragen. Das Ergebnis ist jeweils die Voreinstellung des Registers, wenn man es nicht mit SET angesprochen hat, oder der Wert, den man ihm irgendwann mit SET zugewiesen hat.

SYSTEM-RESET

Die SYSTEM-RESET-Taste dient dazu, laufende Programme zu unterbrechen. Dabei wird der Bildschirm gelöscht, der normale Textmodus aktiviert, die Tabulatorstopps, die Farben und der Bildschirmrand auf die Normalwerte zurückgesetzt, die Tonerzeugung unterbrochen und sämtliche Ein- und Ausgabekanäle geschlossen, wobei es eventuell zum Verlust von Daten kommen könnte.

- Auf den Modellen 400 und 800 kann es passieren, daß der Rechner „abstürzt", d. h. keine Funktion mehr zeigt und nicht auf das Drücken der RESET-Taste reagiert. Dies ist auf einen Fehler im Anschluß dieser Taste zurückzuführen. Die einzige verbleibende Möglichkeit, den Rechner dann wieder zum Leben zu erwecken, ist das Aus- und wieder Einschalten.
- Da die RESET-Taste beim Atari eine echte RESET-Funktion des Mikroprozessors darstellt, gibt es keine Möglichkeit, sie zu „sperren". Allerdings kann man bestimmen, was der Rechner machen soll, nachdem er die genannten Initialisierungsvorgänge beendet hat. Ein solches Programm muß allerdings in Maschinensprache geschrieben werden.

T

TAB (BASIC XL)

Form: TAB (#nausdr1,)nausdr2
nausdr1: Kanalnummer
nausdr2: Nummer der Spalte
Beispiel: TAB 2

Der Befehl TAB dient dazu, auf einem Peripheriegerät (wenn keine Kanalnummer angegeben ist, wird der Bildschirm benutzt) so viele Leerzeichen auszugeben, bis der Anfang der durch nausdr2 spezifizierten Spalte erreicht ist.

Tabulator

Auf der Tastatur Ihres Atari-Computers befindet sich eine Taste mit der Aufschrift „TAB“. Sie dient dazu, den Cursor auf eine bestimmte Position in der laufenden oder in der nächsten Zeile zu setzen (meist auf den Anfang einer neuen Spalte). Die Voreinstellung der Tabulatorpositionen kann durch CLR-TAB (Control und TAB gleichzeitig drücken) gelöscht werden. Neue Positionen können dann folgendermaßen festgelegt werden: Der Cursor wird an die gewünschte Position bewegt und dann SET-TAB (durch gleichzeitiges Drücken von SHIFT und TAB) betätigt.

TAN

Form: TAN (nausdr)
Beispiel: DEG:?TAN(45)

Mit der Funktion TAN kann der Tangens berechnet werden. Je

nachdem, in welchem Rechenmodus sich der Rechner befindet (→ DEG, RAD), muß das Argument in Grad oder Bogenmaß angegeben werden.

Teilzeichenketten

Form: zvar(nausdr1(,nausdr2))
nausdr1: Erstes Zeichen in der Teilzeichenkette
nausdr2: Letztes Zeichen in der Teilzeichenkette
Beispiel: DATA$(5,10)

Aus Zeichenketten können auch Teile „herausgenommen" und einzeln bearbeitet werden. Dazu gibt man hinter der Zeichenkettenvariablen einfach in Klammern an, dem wievielten Zeichen die Teilzeichenkette beginnen soll. Zusätzlich darf, durch ein Komma von der ersten Zeichenkette getrennt, auch das Ende der Teilzeichenkette festgelegt werden. Ein Beispiel:

```
DIM TITEL$(40)
TITEL$="Atari BASIC-Handbuch"
```

Der Ausdruck „TITEL$(7)" würde dann das Wort „BASIC-Handbuch" enthalten.

„TITEL$(13,16)" wäre dementsprechend der Zeichenkettenausdruck „Hand".

Selbstverständlich darf keiner der beiden Parameter die tatsächliche Länge der Zeichenkette überschreiten.

Einem Teil einer Zeichenkette kann auch ein neuer Wert zugeweisen werden, und zwar mit der Anweisung:

TITEL$(1,5)="SYBEX" würde der Zeichenkettenvariable TITEL$ der neue Inhalt „SYBEX BASIC-Handbuch" zugewiesen.

Beachten Sie, daß in diesem Fall auch mittels der zweiten Zahl das Ende der Teilzeichenkette festgelegt werden muß, weil sonst der restliche Teil der Zeichenkette gelöscht würde.

TIME (MSB)

Form: TIME
Beispiel: IF TIME=19999 THEN PRINT "SCHLUSS"

Die Funktion TIME gibt an, wieviel Zeit seit dem Einschalten oder dem Zurücksetzen der Uhr vergangen ist.

TIME$ (MSB)

Form: TIME$
Beispiel: A$=TIME$

Die Zeichenkettenvariable TIME$ arbeitet als eingebaute Digitaluhr mit dem Format „hh:mm:ss“. Genauso wie TIME gibt sie die Zeit seit Einschalten des Computers an, wenn sie nicht seitdem verändert worden ist.

TO

Siehe unter FOR-TO-NEXT-STEP.

TRACE (BASIC XL)

Form: TRACE
Beispiel: TRACE

TRACE schaltet den sogenannten TRACE-Modus ein. Bei eingeschaltetem TRACE-Modus wird parallel zum laufenden Programm ständig die Nummer der aktuellen Programmzeile, d. h. der Zeile, in der sich das Programm gerade befindet, in eckigen Klammern auf dem Bildschirm ausgegeben (siehe TRON in Microsoft-BASIC 2).

TRACEOFF (BASIC XL)

Form: TRACEOFF
Beispiel: TRACEOFF

TRACEOFF schaltet den TRACE-Modus von BASIC XL wieder aus.

TRAP (Atari, BASIC XL)

Abkürzung: TR.
Form: TRAP nausdr
nausdr: anzuspringende Zeilennummer
Beispiel: TRAP 32000

Mit Hilfe des Befehls TRAP kann man den Rechner dazu veranlassen, im Falle eines auftretenden Fehlers, das Programm an einer bestimmten anderen Stelle fortzusetzen. Als Argument gibt man dazu einfach nur die Zeilennummer an, bei der das Programm dann nach Auftreten des Fehlers fortgesetzt werden soll.

Zur Standardbehandlung von Fehlern kann man durch „TRAP 40000“ zurückschalten.

Anmerkungen

- Der Befehl TRAP ist immer nur für den nächsten auftretenden Fehler gültig. Ist es jedoch erwünscht, daß stets die gleiche Fehlerbehandlungsroutine benutzt wird, sollte man den TRAP-Befehl einfach innerhalb dieser Routine wiederholen.
- Es sollte vermieden werden, den TRAP-Befehl in Programmen zu verwenden, die noch nicht vollständig ausgetestet sind. Sonst kann es passieren, daß für den Verlauf des Programms entscheidende Fehler nicht erkannt werden können und dadurch die Fehlersuche entscheidend schwieriger wird.

TROFF (MSB)

Form: TROFF
Beispiel: TROFF

Mit TROFF wird der TRACE-Modus wieder ausgeschaltet (siehe TRACEOFF in BASIC XL).

TRON (MSB)

Form: TRON
Beispiel: TRON

Schaltet den TRACE-Modus ein. Im TRACE-Modus wird ständig die laufende Zeilennummer ausgegeben, um durch genaue Beobachtung des Programmablaufs Fehler feststellen zu können (siehe TRACE in BASIC XL).



UNLOCK (MSB)

Form: UNLOCK zausdr
zausdr: Dateispezifikation
Beispiel: UNLOCK "D:*.*"

Mit UNLOCK kann man die durch LOCK (nicht durch SAVE ...LOCK!) gesicherten Dateien wieder zugänglich machen (siehe UNPROTECT in BASIC XL).

UNPROTECT (BASIC XL)

Abkürzung: UNP.
Form: UNPROTECT zausdr
zausdr: Dateispezifikation einer Diskettendatei
Beispiel: UNPROTECT "D:*.*"

Entspricht dem DOS-Befehl UNLOCK (schaltet den durch PROTECT errichteten Schutz wieder ab).

In Microsoft-BASIC heißt der entsprechende Befehl UNLOCK, in Atari-BASIC kann man sich mit einem XIO-Kommando behelfen.

Unterprogramm

Ein Unterprogramm ist ein Block von Anweisungen, der innerhalb eines Programms mehrfach gebraucht wird.

Wenn man einen solchen Block von Anweisungen mit RETURN abschließt, kann man ihn von jeder beliebigen anderen Stelle im Pro-

gramm mit GOSUB aufrufen. Nach Ausführung der Befehle im Unterprogramm wird die Programmausführung dann automatisch an die Anweisung nach dem aufrufenden GOSUB übertragen.

USR (Atari, BASIC XL)

Form: USR (nausdr1 (,nausdr...))
nausdr1: Anfangsadresse des Maschinensprachprogramms im Speicher
sämtliche folgende nausdr: Parameter, deren Bedeutung jeweils verschieden ist

Beispiel: ? USR(ADR(CIO$),1,ASC("G"),
PEEK(88)*256*PEEK(89),7680)

Die Funktion USR dient zum Aufruf eines Maschinensprachprogramms an der Stelle im Speicher, die durch den ersten Parameter festgelegt worden ist.

Maschinensprachroutinen werden häufig in Zeichenketten abgespeichert, da sie sich so am einfachsten verwenden und übertragen lassen, keine Initialisierungszeit benötigen und sich der Inhalt von Zeichenketten nicht unerwartet ändern kann.

Als Ergebnis liefert die USR-Routine den Inhalt der Doppelbytespeicherzelle 212 zurück. Hier kann man also etwaige Ergebnisse abspeichern und damit ins BASIC-Programm übertragen.

Die interne Funktionsweise: Zunächst wird die Anzahl der Parameter bestimmt. Dabei wird der erste, der ja die Anfangsadresse angegeben hat, nicht berücksichtigt. Da der Wert jedes Parameters zwischen 0 und 65535 liegen kann und deshalb in zwei Bytes abgespeichert werden muß, wird dieser Wert nochmals verdoppelt. Das Ergebnis dieser Berechnung und alle Parameter werden auf den STACK abgelegt und stehen somit dem Unterprogramm zur Verfügung. Da das Unterprogramm durch den Maschinensprachbefehl RTS (Return from Subroutine) beendet wird, muß man vorher sorgfältig alle übergebenen Argumente vom STACK geholt haben, da das Programm sonst bei RTS irgendwo, aber nicht dort, wo es herkommt, hinspringen wird.

USR (MSB)

Form: USR (nausdr,(nausdr))
Beispiel: PRINT USR (CIO,32)

Der USR-Funktion kann in Microsoft-BASIC nur ein Parameter neben der Anfangsadresse übergeben werden.



VAL

Form: VAL(zausdr)
Beispiel: VAL("−3.1")
VAL(Z$)

Mit der Funktion VAL können Zeichenketten, die eine Zahl in ATASCII-Form enthalten, in Zahlen umgewandelt werden. Allerdings darf innerhalb der Zeichenkette eben nur eine Zahl, aber kein numerischer Ausdruck stehen.

- Den umgekehrten Weg der Zahlenumwandlung kann man mit der Funktion STR$ durchführen.

VARPTR (MSB)

Form: VARPTR (var)
Beispiel: ADR=VARPTR(DAT$)

VARPTR erfüllt die gleiche Funktion wie der Atari-BASIC-Befehl ADR. Darüber hinaus kann man auch mit VARPTR die Adresse eines durch „OPTION ausdr" reservierten Speicherbereichs berechnen.

VERIFY (MSB)

Form: VERIFY zausdr
zausdr: Gerätespezifikation
Beispiel: VERIFY "C:"

VERIFY vergleicht die durch zausdr angegebene Datei mit dem im Speicher befindlichen BASIC-Programm. Sollte ein Unterschied festgestellt werden, erscheint eine Fehlermeldung.

Verkettung

Einzelne BASIC-Programme lassen durch Anwendung des Befehls „RUN dateinamen“ verketten.

Möchte man, daß das folgende Programm von Kassette nachgeladen wird, muß man beachten, daß zur Ausführung aller I/O-Operationen mit dem Kassettenrecorder ein Tastendruck notwendig ist. Diesen kann man aber durch „POKE 764,12“ simulieren, so daß das folgende Programm wirklich automatisch nachgeladen werden kann.



WAIT (MSB)

Form: WAIT nausdr1,nausdr2,nausdr3
Beispiel: POKE 764,0:WAIT 764,128,128

Die Programmausführung wird unterbrochen, bis der Wert der Adresse „numausdr1 AND numausdr2“ mit „numausdr3“ übereinstimmt. Bei dem obenstehenden Beispiel würde die Programmausführung erst dann fortgesetzt, wenn eine Tastenkombination mit CONTROL gedrückt wird (Bit 7 gesetzt).

Wild-Cards

Die sogenannten „Wild-Cards“ ermöglichen es, bei Namen von Diskettendateien bestimmte Teile wegzulassen und dadurch viel Arbeit zu sparen. Dazu stehen zwei Sonderzeichen zur Verfügung:

Das Fragezeichen kann irgendein anderes beliebiges Zeichen ersetzen. Bei Angabe von „D:?B??????.???“ würde also auf die erste (bzw. alle) Diskettendateien zugegriffen, dessen zweiter Buchstabe ein „B“ ist.

Das zweite Sonderzeichen, das Sternchen (Multiplikationszeichen), ist in manchen Fällen praktischer. Es ersetzt eine beliebige Zeichenfolge. Gibt man hinter dem Sternchen noch weitere Zeichen ein, dann werden diese folglich auch ignoriert. Da das DOS aber strikt zwischen Namen und Namenserweiterung unterscheidet, sind Dateinamen wie "D:*.BAS“ (das erste Programm mit der Namenserweiterung „.BAS“) dennoch sinnvoll.

WHILE (BASIC XL)

Abkürzung: WH.
Form: WHILE logausdr
...
ENDWHILE
Beispiel:

```
100 WHILE A<0
110 A=A+1
120 ENDWHILE
```

Das Kommando WHILE ermöglicht es, einen Programmteil (der mit ENDWHILE wieder beendet werden muß) so lange auszuführen, wie eine bestimmte Bedingung, die hinter WHILE angegeben werden muß, erfüllt ist.



XIO

Abkürzung: X.
Form: XIO nausdr1, #nausdr2, nausdr3, nausdr4, zausdr
nausdr1: Kommandonummer
nausdr2: Kanalnummer
nausdr3/4: Spezielle Parameter (Kommndoabhängig)
zausdr: Gerätespezifikation
Beispiel: XIO 254,#1,0,0,"D1:*.*"

Mit dem Befehl XIO können sämtliche Ein- und Ausgabeoperationen, die über die CIO (ROM-Routinen für Ein- und Ausgabe) laufen, durchgeführt werden.

Für einige der durch XIO verfügbaren Funktionen stehen allerdings auch eigene BASIC-Befehle zur Verfügung.

Im folgenden eine Liste der zur Verfügung stehenden Kommandos:

Kommando-nummer	Operation Beispiel
3	OPEN
5	Get Record
7	Get Characters
9	Put Record
11	Put Characters
12	CLOSE
13	STATUS
17	DRAW

(Die obenstehenden Kommandos entsprechen exakt den BASIC-Befehlen OPEN, INPUT, GET, PRINT, PUT, CLOSE, STATUS und DRAW)

18	Fill (siehe unten)
32	Rename XIO 32,#1,0,0,"D:*.CAR,*.BAS" (siehe auch unter RENAME)
33	Delete XIO 33,#1,0,0,"D:*.BAS" (siehe auch unter DELETE)
35	Lock XIO 35,#1,0,0,"D:*.BAS" (siehe auch unter PROTECT)
36	Unlock XIO 36,#1,0,0,"D:*.BAS"
37	Point (siehe POINT)
38	NOTE (siehe NOTE)
254	Format XIO 254,#1,0,0,"D:*.*" (formatiert Diskette in Laufwerk 1)

Für den Betrieb eines jeden Peripheriegeräts ist ein Ansteuerungsprogramm, der sogenannte Handler. Für Bildschirm, Tastatur, Drucker und Kassettenrecorder sind die betreffenden Handler bereits im Betriebssystem des Atari fest installiert. Eine Ausnahme bildet das Diskettenlaufwerk, dessen Handler im DOS enthalten ist. Die verschiedene XIO-Kommandos werden jeweils von dem entsprechenden Handler ausgeführt. Deshalb sind für andere Handler als den oben aufgeführten auch andere Kommandonummern möglich.

Die Kanalnummer kann zwischen 1 und 7 liegen, wobei allerdings darauf geachtet werden muß, daß der betreffende Kanal tatsächlich benutzt wird, und daher nicht gleichzeitig für andere Zwecke benutzt werden kann.

Mit den folgenden zwei Parametern können bestimmte Funktionsmodi ausgewählt werden. Bei den XIO-Kommandos, die nicht auch durch einen BASIC-Befehl ausgeführt werden können, müssen diese beiden Zahlen stets auf 0 gesetzt werden.

Mit dem XIO-Kommando 17 können Flächen in einer bestimmten Farbe ausgefüllt werden. Dazu ist folgendermaßen vorzugehen:

1. Die untere rechte Ecke mit PLOT setzen.
2. Mit DRAW zur oberen rechten Ecke eine Linie ziehen.
3. Mit DRAW zur oberen linken Ecke eine Linie ziehen.

4. Mit POSITION den Graphikcursor auf die untere linke Ecke der auszufüllenden Fläche setzen.
5. Mit POKE in der Adresse 756 die entsprechende Farbe ablegen (z. B. POKE 765,1).
6. XIO 18,#6,0,0,"S:"

Stichwortverzeichnis

Die SYBEX Bibliothek

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3005** (1982)

PROGRAMMIERUNG DES Z80
von Rodnay Zaks – ein kompletter Lehrgang in der Programmierung des Z80 Mikroprozessors und eine gründliche Einführung in die Maschinensprache. 608 Seiten, 176 Abbildungen, Format DIN A5, Best.-Nr.: **3006** (1982)

PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3007** (1982)

BASIC COMPUTER SPIELE/Band 1
herausgegeben von David H. Ahl – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Best.-Nr. **3009**

BASIC COMPUTER SPIELE/Band 2
herausgegeben von David H. Ahl – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Best.-Nr.: **3010**

PROGRAMMIERUNG DES 6502 (2. überarbeitete Ausgabe)
von Rodnay Zaks – Programmierung in Maschinensprache mit dem Mikroprozessor 6502, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 368 Seiten, 160 Abbildungen, Format DIN A5, Best.-Nr.: **3011** (1982)

MIKROPROZESSOR INTERFACE TECHNIKEN (3. überarbeitete Ausgabe)
von Rodnay Zaks/Austin Lesea – Hardware- und Software-Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 432 Seiten, 400 Abbildungen, Format DIN A5, Best.-Nr.: **3012** (1982)

VORSICHT! Computer brauchen Pflege
von Rodnay Zaks – das Buch, das Ihnen die Handhabung eines Computersystems erklärt – vor allem, was Sie damit nicht machen sollten. Allgemeingültige Regeln für die pflegliche Behandlung Ihres Systems. 240 Seiten, 96 Abbildungen, Best.-Nr.: **3013** (1983)

6502 ANWENDUNGEN
von Rodnay Zaks – das Eingabe-/Ausgabe-Buch für Ihren 6502-Microprozessor. Stellt die meistgenutzten Programme und die dafür notwendigen Hardware-Komponenten vor. 288 Seiten, 213 Abbildungen, Best.-Nr.: **3014** (1983)

BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

BASIC ÜBUNGEN FÜR DEN APPLE
von J.-P. Lamoitier – das Buch für APPLE-Nutzer, die einen schnellen Zugang zur Programmierung in BASIC suchen. Abgestufte Übungen mit zunehmendem Schwierigkeitsgrad. 256 Seiten, 190 Abbildungen, Best.-Nr.: **3016** (1983)

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 576 Seiten, 325 Abbildungen, Best.-Nr.: **3017** (1984)

EINFÜHRUNG IN DIE TEXTVERARBEITUNG
von Hal Glatzer – woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 248 Seiten, 67 Abbildungen, Best.-Nr. **3018** (1983)

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

MEIN SINCLAIR ZX81
von D. Hergert – eine gut lesbare Einführung in diesen Einplatinencomputer und dessen Programmierung in BASIC. 176 Seiten, 47 Abbildungen, Best.-Nr.: **3021** (1983)

SINCLAIR ZX SPECTRUM Programme zum Lernen und Spielen
von T. Hartnell – ein Buch zur praktischen Anwendung. Grundzüge des Programmierens aus dem kaufmännischen Bereich sowie Spiele, Lehr- und Lernprogramme in BASIC. 232 Seiten, 140 Abbildungen, Best.-Nr. **3022** (1983)

BASIC ÜBUNGEN FÜR DEN IBM PERSONAL COMPUTER
von J.-P. Lamoitier – vermittelt Ihnen BASIC durch praktische und umfassende Übungen anhand von realistischen Programmen: Datenverarbeitung, Statistik, kommerzielle Programme, Spiele u.v.m. 256 Seiten, 192 Abbildungen, Best.-Nr.: **3023** (1983)

PROGRAMMSAMMLUNG ZUM IBM PERSONAL COMPUTER
von S. R. Trost – mehr als 65 getestete, direkt einzugebende Anwenderprogramme, die eine weite Palette von kaufmännischen, persönlichen und schulischen Anwendungen abdecken. 192 Seiten, 158 Abbildungen, Best.-Nr.: **3024** (1983)

PLANEN UND ENTSCHEIDEN MIT BASIC
von X. T. Bui – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

BASIC FÜR DEN KAUFMANN
von D. Hergert – das BASIC-Buch für Studenten und Praktiker im kaufmännischen Bereich. Enthält Anwendungsbeispiele für Verkaufs- und Finanzberichte, Grafiken, Abschreibungen u.v.m. 208 Seiten, 85 Abbildungen, Best.-Nr.: **3026** (1983)

SINCLAIR ZX SPECTRUM BASIC HANDBUCH
von D. Hergert – eine wichtige Hilfe für jeden SPECTRUM-Anwender. Gibt eine Übersicht aller BASIC-Begriffe, die auf diesem Rechner verwendet werden können, und erläutert sie ausführlich anhand von Beispielen. 288 Seiten, 188 Abbildungen, Best.-Nr.: **3027** (1983)

SINCLAIR ZX81 BASIC HANDBUCH
von D. Hergert – vermittelt Ihnen das vollständige BASIC-Vokabular anhand von praktischen Beispielen, macht Sie zum Programmierer Ihres ZX81. 181 Seiten, 120 Abbildungen, Best.-Nr.: **3028** (1983)

PROGRAMME FÜR MEINEN APPLE II
von S. R. Trost – enthält eine Reihe von lauffähigen Programmen samt Listing und Beispiellauf. Hilft Ihnen, viele neue Anwendungen für Ihren APPLE II zu entdecken und erfolgreich einzusetzen. 192 Seiten, 158 Abbildungen, Best.-Nr.: **3029** (1983)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

APPLE II LEICHT GEMACHT
von J. Kascmer – macht Sie schnell mit Tastatur, Bildschirm und Diskettenlaufwerken vertraut. Sie lernen, wie leicht es ist, Ihr eigenes BASIC-Programm zu schreiben. 192 Seiten, mit 43 Abbildungen, Best.-Nr.: **3031** (1984)

PLANEN, KALKULIEREN, KONTROLLIEREN MIT BASIC-TASCHENRECHNERN
von P. Ickenroth – präsentiert eine Reihe von direkt anwendbaren BASIC-Programmen für zahlreiche kaufmännische Berechnungen mit Ihrem BASIC-Taschenrechner. 144 Seiten, 48 Abbildungen, Best.-Nr.: **3032** (1983)

MEIN ERSTES BASIC PROGRAMM
von Rodnay Zaks – das Buch für Einsteiger! Viele farbige Illustrationen und leichtverständliche Diagramme bringen Spaß am Lernen. In wenigen Stunden schreiben Sie Ihr erstes nützliches Programm. 208 Seiten, illustriert, Best.-Nr.: **3033** (1983)

IBM PC-DOS HANDBUCH
von R. A. King – umfassende Einführung in das Disketten-Betriebssystem Ihres IBM PC, seine grundsätzlichen Möglichkeiten und Funktionen sowie auch fortgeschrittene Funktionen (einschließlich der Version 2.0). 320 Seiten, ca. 50 Abbildungen, Best.-Nr.: **3034** (1984)

APPLE II BASIC HANDBUCH
von D. Hergert – ein handliches Nachschlagewerk, das neben Ihren Apple II, II+ oder IIe stehen sollte. Dank vieler Tips und Vorschläge eine wesentliche Erleichterung fürs Programmieren. 304 Seiten, 116 Abbildungen, Best.-Nr. **3036** (1984)

Z80 ANWENDUNGEN
von J. W. Coffron – vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren. 296 Seiten, 204 Abbildungen, Best.-Nr.: **3037** (1984)

COMMODORE 64 – LEICHT GEMACHT
von J. Kascmer – führt Sie schnell in die Bedienung von Tastatur, Bildschirm und Diskettenlaufwerken ein, macht Sie zum BASIC-Programmierer Ihres C64! 176 Seiten, 36 Abbildungen, Best.-Nr.: **3038** (1984)

SPIELEN, LERNEN, ARBEITEN mit dem TI99/4A
von K.-J. Schmidt und G.-P. Raabe – eine eingehende Einführung in die Bedienung und Programmierung des TI99/4A. Mit den vielen Beispielprogrammen holen Sie das Beste aus Ihrem Computer heraus. 192 Seiten, 41 Abbildungen, Best.-Nr.: **3039** (1984)

MEIN ERSTER COMPUTER
von Rodnay Zaks – Der unentbehrliche Wegweiser für jeden, der den Kauf oder den Gebrauch eines Mikrocomputers erwägt, das Standardwerk in 3., überarbeiteter Ausgabe. 304 Seiten, 150 Abbildungen, zahlreiche Illustrationen, Best.-Nr.: **3040** (1984)

MEIN DRAGON 32
von N. Hesselmann – entwickelt Ihre Fähigkeiten in der Nutzung, Programmierung und erweiterten Anwendung Ihres Rechners anhand von vielen Beispielprogrammen. 256 Seiten, 41 Abbildungen, Best.-Nr.: **3041** (1984)

ERFOLG MIT MULTIPLAN
von Th. Ritter – das Tabellenkalkulations-Programm Multiplan hilft Ihnen bei der Lösung kommerzieller, wissenschaftlicher und allgemeiner Probleme. Lernen Sie die Möglichkeiten kennen, Ihre Software optimal zu nutzen! 208 Seiten, ca. 60 Abbildungen, Best.-Nr.: **3043** (1984)

FARBSPIELE MIT DEM COMMODORE 64
von W. Black und M. Richter – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

FORTGESCHRITTENE 6502-PROGRAMMIERUNG
von Rodnay Zaks – hilft Ihnen, schwierige Probleme mit dem 6502 zu lösen, stellt Ihnen Maschinenroutinen zum Arbeiten mit einem Hobbyboard vor. 288 Seiten, 140 Abbildungen, Best.-Nr.: **3047** (1984)

COMMODORE 64 BASIC HANDBUCH
von D. Hergert – zeigt Ihnen alle Anwendungsmöglichkeiten Ihres C64 und beschreibt das vollständige BASIC-Vokabular anhand von praktischen Beispielen. 208 Seiten, 92 Abbildungen, Best.-Nr.: **3048** (1984)

PROGRAMMIERUNG DES 6809
von R. Zaks und W. Labiak – eine vollständige Einführung in die Assemblerprogrammierung mit dem 6809, für alle, die mit DRAGON 32, Tandy Colorcomputer oder einem anderen 6809-System arbeiten. 400 Seiten, 150 Abbildungen, Best.-Nr.: **3049** (1984)

PROGRAMMIERUNG DES 8086/8088
von J. W. Coffron – lehrt Sie Programmierung, Kontrolle und Anwendung dieses 16-Bit-Mikroprozessors; vermittelt Ihnen das notwendige Wissen zu optimaler Nutzung Ihrer Maschine, von der internen Architektur bis hin zu fortgeschrittenen Adressierungstechniken. 312 Seiten, mit Abbildungen, Best.-Nr.: **3050** (1984)

COMMODORE 64 PROGRAMMSAMMLUNG
von S. R. Trost – mehr als 70 getestete Anwenderprogramme, die direkt eingegeben werden können. Erläuterungen gewährleisten eine optimale Nutzung. 192 Seiten, 160 Abbildungen, Best.-Nr.: **3051** (1983)

CP/M-HANDBUCH
von Rodnay Zaks – das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M. 2., überarbeitete Ausgabe. 320 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

UNIX-HANDBUCH
von R. Detering – eine systematische Einführung in UNIX, das kommende Betriebssystem für 16-bit-Rechner. Lernen Sie, Ihren Prozessor optimal einzusetzen! Ca. 280 Seiten, ca. 30 Abbildungen, Best.-Nr.: **3054** (1984)

ERFOLGREICH PROGRAMMIEREN MIT C
von J. A. Illik – ein unentbehrliches Handbuch für jeden, der mit der universellen Sprache C erfolgreich programmieren will. Aussagekräftige Beispiele, auf verschiedenen Mini- und Mikrocomputern getestet. Ca. 400 Seiten, Best.-Nr.: **3055** (1984)

ARBEITEN MIT DEM IBM PC
von J. Lasselle und C. Ramsay – zeigt Ihnen Schritt für Schritt, wie Sie den IBM PC ohne Vorkenntnisse einsetzen, die speziellen Eigenschaften dieses Computers für Druck, Grafik und Kommunikation nutzen können. 156 Seiten, ca. 30 Abbildungen, Best.-Nr.: **3056** (1984)

MEIN ERSTES COMMODORE 64-PROGRAMM
von R. Zaks – sollte Ihr erstes Buch zum Commodore 64 sein. Viel Spaß am Lernen durch farbige Illustrationen und leichtverständliche Diagramme, Programmieren mit sofortigen Resultaten. 208 Seiten, illustriert, Best.-Nr.·**3062** (1984)

SYBEX MIKROCOMPUTER LEXIKON
– die schnelle Informationsbörse! Über 1500 Definitionen, Kurzformeln, Begriffsschema der Mikroprozessor-Technik, englisch/deutsches und französisch/deutsches Wörterbuch, Bezugsquellen. 192 Seiten, Format 12,5 x 18 cm, Best.-Nr.: **3035** (1984)

COMPUTER TOTAL VERRÜCKT
von Daniel Le Noury – mit diesem Buch kommen Sie wieder zur Besinnung, nachdem Sie sich halbtot gelacht haben. Ca. 100 Cartoons rund um den Computer. 96 Seiten, Best.-Nr. **3042** (1984)

MEIN COLOUR GENIE
von Ralf Marquis – zeigt Ihnen, wie man mit einfachen Mitteln eindrucksvolle Programme für den Colour-Genie erstellen kann; viele praktische Beispiele. 160 Seiten, 76 Abb., Best.-Nr. **3063** (1984)

MEIN HEIMCOMPUTER
von N. Hesselmann – zeigt, was ein Heimcomputer ist und was man mit ihm anfangen kann, von den Chips bis zu Tips für den Kauf. Ca. 250 Seiten, ca. 100 Abb., Best.-Nr. **3064** (1984)

MIT DEM COMPUTER UNTERWEGS
von W. Höfs – für alle, die einen netzunabhängigen Rechner benötigen; alles über Handheld-Computer. Ca. 200 S., mit Abb., Best.-Nr. **3067** (November 1984)

ATARI PROGRAMM-SAMMLUNG
von S. R. Trost – sollte neben keinem Atari-Computer fehlen. Es bietet einen Satz ausgetesteter Programme für eine Fülle von Anwendungen. 190 S., 150 Abb., Best.-Nr. **3068** (1984)

PLANEN + ENTSCHEIDEN MIT DEM SHARP PC-1500
von X. T. Bui/H. Klein – eine Sammlung interaktiver, kommerziell orientierter BASIC-Programme für Analysen, Planung und Prognosen. Auch für Tandy PC-2. 224 Seiten, 50 Abb., Best.-Nr. **3069** (1984)

ARBEITEN MIT dBase II
von A. Simpson – Grundlagen und Programmiertechniken für die Datenbank-Verwaltung mit dBASE II. Zahlreiche praktische Tips. Ca. 240 Seiten, ca. 50 Abb., Best.-Nr. **3070** (1984)

SPASS AN MATHE MIT DEM COMMODORE 64
von H. Danielsson – zeigt Ihnen mit vielen Beispielen, wie der C 64 für schulische oder private Berechnungen genutzt werden kann. Ca. 250 Seiten mit Abb., Best.-Nr. **3072** (1985)

COMMODORE 64 – GRAFIK + DESIGN
von Ch. Platt – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. Ca. 280 S., ca. 150 Abb., teils vierfarbig. Best.-Nr. **3073** (1984)

SVI PROGRAMM-SAMMLUNG
von S. R. Trost – Knapp 70 ausgetestete Anwenderprograme, u. a. für kommerzielle Berechnungen, Dateiverwaltung und mathematische Übungen; ohne Vorkenntnisse nutzbar. 192 Seiten, 160 Abb., Best.-Nr. **3074** (1984)

V24/RS-232 KOMMUNIKATION
von J. Campbell – zeigt Ihnen, wie Sie mit den Schnittstellen V24 und RS-232 notwendiges Zubehör an Ihren Rechner anschließen; mit praktischen Fallbeispielen. Ca. 200 Seiten, 97 Abb., Best.-Nr. **3075** (1984)

PROGRAMMIEREN MIT CP/M
von A. R. Miller – vermittelt die Feinheiten von CP/M und hilft, die Möglichkeiten dieses populären Betriebssystems zu erweitern. Ca. 450 Seiten, ca. 100 Abb., Best.-Nr. **3077** (1984)

ARBEITEN MIT DEM MACINTOSH
von N. Hesselmann – alles über den leistungsfähigen Apple-Rechner mit einer Erläuterung wichtiger kommerzieller Software-Pakete und deren Einsatz, Anleitung zur Programmierung in Microsoft-BASIC. Viele konkrete Anwendungs-Beispiele. Ca. 400 Seiten, zahlr. Abb., Best.-Nr. **3080** (1984)

alphatronic PC BASIC HANDBUCH
von K.-H. Hauer – erläutert das vollständige BASIC-Vokabular für den alphatronic PC mit vielen direkt einsetzbaren Beispiel-Programmen. Ca. 250 Seiten, mit Abb., Best.-Nr. **3084** (November 1984)

MEIN ZWEITES COMMODORE 64 PROGRAMM
von Gary Lippman – für alle, die bereits ein Grundwissen in BASIC haben und mit ihrem C 64 den nächsten Schritt machen wollen – und das mit viel Spaß. Ca. 250 Seiten, zahlr. witzige Illustr., Best.-Nr. **3086** (1984)



Fordern Sie ein Gesamtverzeichnis unserer Verlagsproduktion an:

SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (0211) 626441
Telex: 8588163

SYBEX INC
2344 Sixth Street
Berkeley, CA 94710, USA
Tel.: (415) 848-8233
Telex: 336311

SYBEX Ltd.
Unit 4, Bourne Industrial Estate
Crayford, Kent DA1 4BU
Tel.: Crayford (0322) 57717
Telex: 897958

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX

ATARI BASIC Handbuch

Das ABC der BASIC-Programmierung für Ihren Atari

BASIC ist nach wie vor die gebräuchlichste Programmiersprache, mit deren Hilfe Sie alle Möglichkeiten Ihres Atari voll ausschöpfen können. Sprechen Sie die Sprache, die Ihr Computer versteht, und erleben Sie seine Leistungsfähigkeit.

Das vorliegende BASIC Handbuch hilft Ihnen, Ihren Atari voll und ganz zu beherrschen. Das vollständige BASIC-Vokabular von A–Z wird beschrieben und anhand praktischer Beispiele erläutert. Mit Hilfe der vielen Beispiel-Programme lernen Sie alle Fähigkeiten Ihres Atari kennen und können nach kurzer Zeit effektive BASIC-Programme für Ihre Zwecke schreiben.

ISBN 3-88745-083-3